

Software Defect Classification

A case study for a core system transformation project

João Rui Machado Costa

Dissertation

FEUP Supervisor: Professor Maria Antónia da Silva Lopes de Carravilla



Master in Industrial Engineering and Management

2016-07-04

Learning implies that every person in every process must have the freedom to observe and identify change and have the power to recommend opportunities for improvement.
— *H. Thomas Johnson*

Resumo

Na indústria de serviços, os sistemas de tecnologias de informação assumem um papel indispensável na área do negócio e representam uma parte substancial da estrutura de custos. A consultora Deloitte foi contratada no sentido de viabilizar a reestruturação do negócio de uma seguradora dinamarquesa, tendo como principal *driver* a reposição do sistema *core*.

No processo de transição de sistema *core*, é necessário implementar novos requisitos, tanto no novo sistema *core* como em outros com os quais interage. Para garantir a qualidade das implementações, cada novo requisito deve ser testado. Esta dissertação pretende criar um sistema standardizado de classificação e de reporte de defeitos de *software* que, com a sua operacionalização, permita aumentar a qualidade dos novos desenvolvimentos do sistema.

Para detetar e compreender os principais pontos a melhorar, foi efetuada uma análise do estado documentado, recorrendo à documentação da Deloitte e do cliente, e do estado real, através da observação no contexto real de trabalho, da análise do histórico de defeitos e de um questionário interno. Foi verificado, por exemplo, que um terço dos defeitos abertos terminam como não resolvidos, pelo facto de não serem efetivamente defeitos. As causas para os principais problemas identificados são diversas e estão relacionadas com algumas falhas de comunicação, as práticas instauradas de classificação e a ausência de papéis e de responsabilidades claramente definidos nas várias atividades associadas às etapas de teste.

Com base nas causas previamente enumeradas, as áreas de melhoria foram identificadas. O modelo *IDEAL* do *Software Engineering Institute* foi utilizado para estruturar a abordagem de melhoria, definindo, para tal, um estado desejado e as mudanças necessárias para o efeito.

Em conclusão, o objetivo proposto foi alcançado, sendo que no futuro, com a implementação do novo sistema de classificação, será expectável que a classificação se torne um processo mais eficiente e eficaz, promovendo melhores resultados de qualidade. Para além disso, com o novo sistema, será possível obter informação relativa à performance do processo com maior facilidade, permitindo identificar as principais falhas a colmatar numa lógica de melhoria contínua.

Abstract

In the services industry, the information technology systems are of critical importance for the business and represent a substantial part of the cost structure. Deloitte, a consultancy firm, has been hired to support a Danish Insurance Company undergoing a major business transformation, whose principal driver is the replacement of the core system.

During the transition process, it is necessary to implement new requirements, not only in the new system level but also in the others with which it operates. To assure the quality of the implementations, every new development must be tested. The objective of this dissertation is to create a software defect classification framework that ultimately promotes improved quality of the system.

To identify improvement areas, an AS IS analysis was performed to define the documented state, based on guidelines from Deloitte and the client, as well as the actual state, through observation, defect historic analysis and a survey conducted to Deloitte's professionals. For instance, it was found that one-third of the defects reported do not end up fixed because they do not constitute actual issues. The root causes of the main gaps identified are related to faulty communication, established classification practices and lack of defined roles and responsibilities within the various activities associated with testing.

The improvement areas were identified based on the previously enumerated root causes. Through the IDEAL model of the Software Engineering Institute, the improvement effort was structured by defining the TO BE model and the required changes to achieve it.

To conclude, the proposed objective has been achieved. In the future, with the implementation of the new classification system, it is expectable an improved efficiency and effectiveness in the defect classification process and consequently improved quality of the software developed. Furthermore, the simplified collection of information concerning processes performance will enable easier identification of the main flaws to tackle in a continuous improvement effort.

Acknowledgment

Firstly, I would like to express my sincere gratitude to my supervisor Professor Maria Carravilla for her attention and guidance as well as to FEUP community for providing me with the resources, the challenges, and the opportunities to become a capable professional.

Secondly, I would like to thank João Filipe Ferreira and Gonalo Moraes, my advisors at Deloitte, for sharing their insights and giving me the opportunity to learn daily from their experience.

I am also grateful to Deloitte, which provided me with valuable knowledge and a vibrant network of excellent professionals, and to my office co-workers who made my adaptation easier and my job meaningful.

Last but not least, I would like to thank everyone who has supported me during throughout my life.

Contents

1	Introduction.....	1
1.1	Context and Motivation.....	1
1.2	Project - Defect Classification at Deloitte	2
1.3	Objectives	3
1.4	Methodology and Planning.....	3
1.5	Structure.....	4
2	Literature Review	5
2.1	Software Life Cycle: V-model	5
2.2	Software Quality.....	6
2.2.1	Software Anomaly	7
2.2.2	Origin of Defects.....	7
2.2.3	Relevance of Defects	7
2.2.4	Costs of Finding and Fixing Software Defects	8
2.3	Testing	8
2.3.1	Responsibilities of the tester.....	8
2.3.2	Risk Based Approach	9
2.4	Defect Reporting and Classification	9
2.4.1	Reporting - Good Practices	9
2.4.2	Classification of Software Defects	9
2.4.3	Defect Rejection	10
2.5	Microsoft Team Foundation Server	11
2.5.1	General Information.....	11
2.5.2	Test Management.....	11
2.5.3	Work Items	11
2.5.4	Source Control – Branching and Merging.....	13
2.6	IT Management Standards.....	13
2.6.1	Process Relevance in Quality.....	13
2.6.2	Capability Maturity Model Integration (CMMI)	14
3	Problem Analysis.....	15
3.1	Requirement Methodology	15
3.2	Testing	15
3.3	AS IS – Documented.....	17
3.3.1	Deloitte Documented Practices	17
3.3.2	Client Documented Practices	22
3.4	AS IS – Actual Practices	22
3.4.1	Observation	23
3.4.2	Defect Historic Analysis.....	24
3.4.3	Survey	28
3.5	Summary.....	31
4	Roadmap for Improvement	32
4.1	Initiating Phase.....	32
4.1.1	Sub phase 1 - Stimulus for Change.....	32
4.1.2	Sub phase 2 - Set Context	33
4.1.3	Sub phase 3 - Build Sponsorship	33
4.1.4	Sub phase 4 - Charter Infrastructure	33
4.2	Diagnosing Phase	33
4.2.1	Sub phase 5 - Characterize Current and Desired State	33
4.2.2	Sub phase 6 - Develop Recommendations	35
4.3	Establishing Phase.....	44
4.3.1	Sub phase 7 - Set Priorities	44

4.3.2 Sub phase 8 - Develop Approach.....	45
4.3.3 Sub phase 9 - Plan Actions	45
4.4 Acting and Learning Phases	45
5 Conclusions and Future Work	46
References	48
APPENDIX A:	50
APPENDIX B:	51
APPENDIX C:	52
APPENDIX D:	53
APPENDIX E:	54
APPENDIX F:	55
APPENDIX G:	61
APPENDIX H:	63
APPENDIX I:	69
APPENDIX J:	74

Glossary

Term	Definition
ALM	Application Life Cycle Management
CMMI	Capability Maturity Model Integration is a process improvement approach whose goal is to help organizations improve their performance
IT	Information Technology
TFS	Team Foundation Server. Microsoft collaboration server for supporting full ALM

List of Figures

Figure 1 – Implementation process overview.....	2
Figure 2 – Plan of activities.....	4
Figure 3 – V-model development methodology (Schaefer 2014)	5
Figure 4 – Relative cost to fix software defects in each phase of its life cycle (Dawson et al. 2010).....	8
Figure 5 – Adapted from TFS test management (Microsoft 2012)	11
Figure 6 – General life cycle for a defect	12
Figure 7 – Critical dimensions for quality improvement (SEI 2010).....	13
Figure 8 – Requirement delivery overview: functional perspective	15
Figure 9 – Testing methodology overview: Deloitte’s perspective.....	16
Figure 10 – Information system architecture: test phase scope.....	17
Figure 11 – Generic defect life cycle.....	19
Figure 12 – A third of the defects reported end up not FIXED.....	25
Figure 13 – Software process improvement framework (SEI 2009).....	32
Figure 14 – Defect fields responsibilities: life cycle overview	37
Figure 15 – Flowchart of activities to follow before opening a defect.....	41

List of Tables

Table 1 – V-model breakdown (Schaefer 2014)	6
Table 2 – Defect attributes: adapted from (IEEE 2010).....	10
Table 3 – Defect description: relevant fields.....	12
Table 4 – RESOLVED REASON nomenclature	18
Table 5 – SEVERITY field criteria.....	20
Table 6 – Root causes per RESOLVED REASON (Historic analysis)	27
Table 7 – Number of occurrences levels	28
Table 8 – Root causes per RESOLVED REASON (Survey).....	29
Table 9 – Percentage levels	29
Table 10 – List of problems identified	31
Table 11 – Current and desired state comparison – CL - 2 MANAGED and CL - 3 DEFINED	34
Table 12 – Roles and responsibilities in testing	35
Table 13 – Work items list: defect and impediment.....	39
Table 14 – RESOLVED REASON: definition and comments	42
Table 15 – Proposed improvements to address faults in defect reporting.....	43
Table 16 – Hierarchy of improvement suggestions.....	44

1 Introduction

The present project was proposed to address part of a business transformation at a Deloitte's client, namely an Insurance Company, driven by a core system¹ replacement. The client is undergoing a major transformation program, restructuring its insurance products and their price.

As a result of the complexity and duration of the project, this transformation is divided into smaller initiatives, organized within releases. Among these initiatives, there is a particular type that focuses on implementing new business requirements² into the already existent information system. To assess the conformity of those implementations with the business requirements, the initiatives end with a testing phase.

During the testing phase, defects³ may arise and thus the need to classify them and offer a resolution.

The result of this dissertation is a scalable framework to report and classify defects, through an improved set of processes and methodologies that deliver increased effectiveness and efficiency in their reporting and classification.

1.1 Context and Motivation

Within a fast changing and unpredictable economic environment, every business must evolve and adapt in order to assure success and longevity. Thus, significant investments are made as to develop and maintain a competitive advantage while keeping the business profitable. The value added outcome potential of technology investments is attractive. Below is cited an example of the business relevance of IT, as to enforce the relevance of the subject.

Application programming interfaces (APIs) have been elevated from a development technique to a business model driver and boardroom consideration. An organization's core assets can be reused, shared, and monetized through APIs that can extend the reach of existing services or provide new revenue streams. APIs should be managed like a product—one built on top of a potentially complex technical footprint that includes legacy and third-party systems and data. (Deloitte 2015, 24)

In the industry of this client, a very large part of the business cost structure is related to information systems and, more importantly, they can exploit data to significantly generate business value (Marr 2015).

¹ Core system – data processing application that runs every IT business related activity. In a business language, it is a system that supports and manages processes across the value chain

² Requirement – a business need or a feature to be implemented into the software system

³ Defect – in software, defect is an imperfection or deficiency in a work product that causes its behaviour to deviate from the expected result (Graham, Van Veenendaal, and Evans 2008)

Moreover, this investment in technology comes from the increased hardship in growing and enhancing profitability (Ryu 2007), hence the need to look for alternative approaches. Modern web-based solutions bring to the market increased ease in the way of doing business, while tackling serious issues in the industry related to the talent retirement, which is considered a key driver in the Insurance industry performance (Deloitte 2006).

From an internal point of view, to keep the innovation and the continuous improvement fueled, it is necessary to adopt changes in the organization. In this case, the project addresses the technological integration of a new core system for the production of insurance services. This new system brings improved flexibility compared to the legacy one, while decreasing the associated run cost.

According to Ryu, one of the founders of Guidewire, a core back-end software for Insurance, no single decision will affect the long-term sustainability to such an extent as the migration of core systems. For that reason, well timed and planned migration can significantly improve the insurer operations without prohibitive enterprise risk, allowing to use operational efficiency as a competitive strength to promote growth and increased industry profitability (Ryu 2007).

To summarize, the Deloitte's client makes considerable investments in its IT department so as to adapt its business to the new digital trends and ensure the long-term success of its operation. In this case, the replacement of a core system requires considerable organization effort and involves a high degree of risk, which creates the need to attend to every detail of the transition.

1.2 Project - Defect Classification at Deloitte

The customer is undergoing a major business transformation, which has the replacement of the core system as its principal driver. Deloitte, considered a Premium Solution Partner by the provider of the core system, was chosen to assist in this process.

The initiative in study consists in the implementation of new business requirements into the information system. As the business changes, its system must also offer new functionalities, not only at the core system service level but also at the other systems it interacts alongside.

This process is presented in Figure 1, starting with high-level planning to clarify the expectations of the solution and its business value, followed by functional design, which assures that the modules of the solution are capable of performing the required tasks. Later, the technical phase of implementation is carried out to materialize the functions previously designed and testing occurs to verify the software quality. Finally, the software system enters production by its integration into the operation of the organization.



Figure 1 – Implementation process overview

In a release, a set of new business requirements is placed under scope with a defined start and end dates. During this time window, Deloitte teams will communicate actively with the client in order to ensure that the steps presented above can be correctly executed and the expectations met.

The problem addressed by this dissertation is the classification of software defects during testing, analyzed from a functional point of view, so as to increase the quality of the system and the ease of managing those issues.

During this phase, each requirement implementation is tested to assure its conformity with the design. When testing reveals defects, these defects must be reported and the tester must register relevant details to guide technical teams⁴ in identifying the cause and solve the problem.

Deloitte, the firm which supported this project, is (...) *the brand under which tens of thousands of dedicated professionals in independent firms throughout the world collaborate to provide audit, consulting, financial advisory, risk management, tax, and related services to select clients.* (Deloitte 2016)

Within the consulting services, Deloitte has a branch dedicated to Financial Services Industry, which consists in Banking and Insurance Companies. Inside this branch, there is a service area called TI, which stands for Technology Integration that tackles technical architecture, systems design, development and integration, system requirements, business intelligence, digital strategy, among other subjects (Deloitte 2014).

From the many member firms composing the brand, this dissertation occurs in the Portuguese member firm Deloitte Consultores S.A., which covers Portugal and Angola.

Most of this member firm activity is within the Financial Services Industry, serving multinational clients. Recently, the firm has been investing in a new digital consulting practice and in new careers to support its IT initiatives, such as Software Engineer. This new development arm in consultancy is critical to build knowledge and the capabilities required to offer excellence in IT related services.

1.3 Objectives

The main objective of this dissertation is a new classification framework with tailored guidelines and process improvement practices. In detail, the key goals are:

- creation of a standardized classification system;
- definition of workflows and responsibilities within teams;
- leaner testing processes;
- improved quality of the system.

1.4 Methodology and Planning

Understanding the current activities within teams from Deloitte and the client related to testing and defect reporting is relevant to improve the performance of defect management procedures. Thus, qualitative information on documented processes models and the real observed teams' workflow will be considered for the purpose of the dissertation.

Also, to complement this approach, a survey among coworkers is included and a quantitative analysis of the defects' historic on previous releases is performed to contextualize the qualitative approach with past reports data.

Through this research method, the subject of improvement is defined and a new model of classifying incidents is delivered to tackle eventual gaps.

⁴ Technical Team – refers to the elements that elaborate the technical design documents and translate the functional Design into code

Concerning the plan of activities, it is presented in the Gantt chart in Figure 2. The main activities are described in the chart associated to a time window so as to keep track of the work progress.

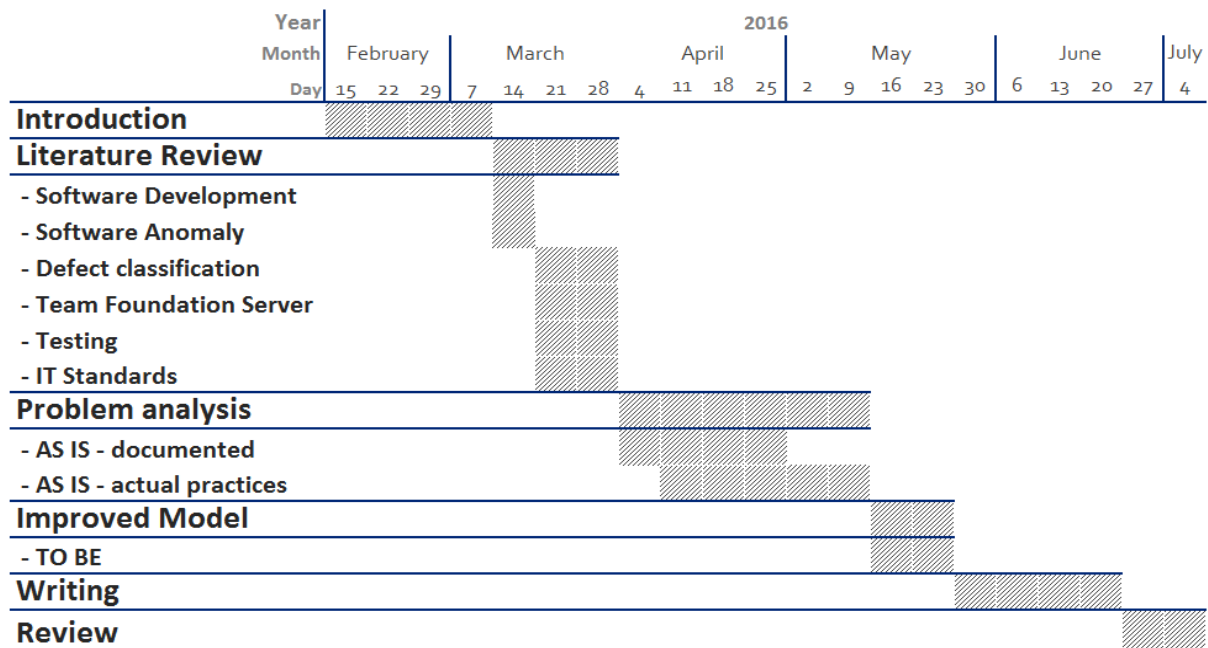


Figure 2 – Plan of activities

1.5 Structure

The present chapter contains an overview of the project, specifically its context and objectives, the company where it took place, the employed methodology and plan, and the structure of the dissertation.

In chapter 2 of this dissertation, literature will be reviewed to build a sound knowledge base of previous work developed to address similar and related issues and compiled the relevant information studied in the respective chapter.

In chapter 3, the problem is analyzed and structured. Both quantitative and qualitative methods will be used to identify the problem and assess its magnitude, as well as field work within Deloitte's and client's teams.

The presentation of the proposed solution follows in chapter 4, tackling the main gaps detected in the problem analysis. Through the standardization of classification and redefinition of roles and responsibilities, those gaps can be mitigated and continuous improvement processes installed.

Finally, in chapter 5 the conclusions and future work is presented. The attainment of the proposed objectives is analyzed, as well as future prospects of this project.

2 Literature Review

The present chapter goal is to deliver background knowledge on the subjects relevant to the dissertation. At first, the contextual software development methodology is presented, followed by the definition of software defect and reporting norms. Since the client makes use of a particular collaborative tool to handle defects, the relevant aspects concerning this tool are presented complemented with some considerations related to testing practices, through which defects are revealed. In the end, a reference to organizational improvement standards is made available.

2.1 Software Life Cycle: V-model

The main idea behind the general V-model is that development and testing tasks are corresponding activities of equal importance. The two branches of the V symbolize this. (Schaefer 2014)

To accomplish a structured and controllable software development practice, development models are employed. Some examples are the Waterfall model and the V-model. The first one is extremely simple and easy to understand but presents testing as a final inspection of the product. Thus, and in analogy to a manufacturing inspection, it prevents quality control to be placed at a process level where it is the most relevant to achieve competitive quality without prohibitive costs. V-model appears as an enhancement of the Waterfall model, embedding testing throughout the whole development process. This development methodology promotes meticulous design, development, and documentation thus should be applied in projects where a rigid structure is required (Schaefer 2014). The V-model model is represented in Figure 3.

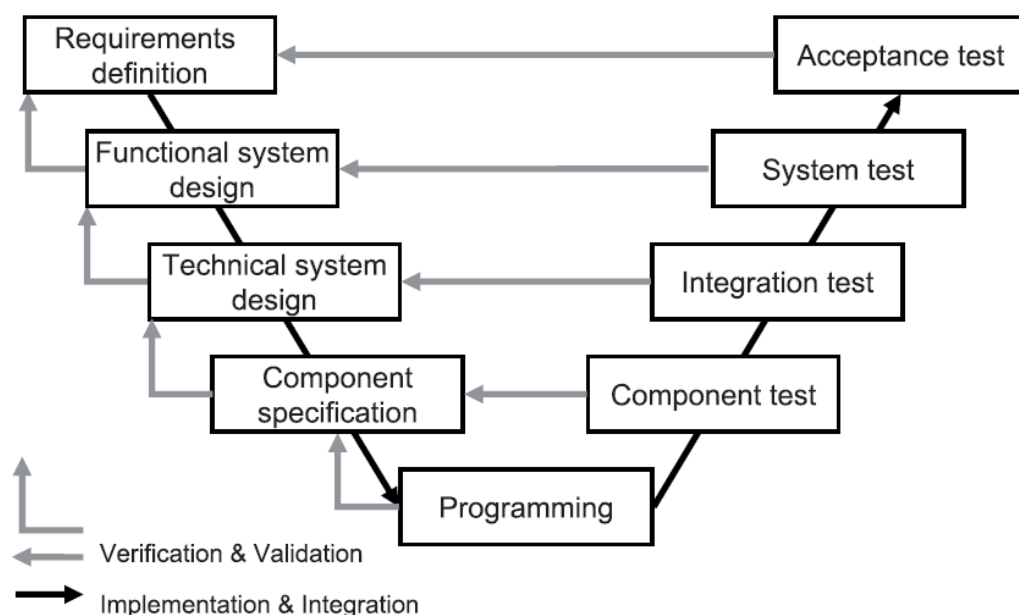


Figure 3 – V-model development methodology (Schaefer 2014)

V-model is the current client's practice for software development, thus the relevance of its study. Note that the left branch relates to the software development and the right one to the associated test levels. In Table 1 a brief description of each step is available.

Table 1 – V-model breakdown (Schaefer 2014)

Requirements definitions	Business needs that a system must support. In this step, the requirements are gathered, specified and approved
Functional system design	Translation of the requirements defined in the previous step into functions and interfaces of the new system
Technical system design	Breakdown of the system architecture into smaller subsystems which should be able to be developed as independently as possible. In other words, it is the design of the implementation
Component specification	Each subsystem task and interface is defined
Programming	Code each component
Component test	Check each individual component
Integration test	Verification of the components in an integrated environment ⁵
System test	Tests if the whole system meets the functional design
Acceptance	Tests if the entire system meets the specified customer requirements

To summarize, each test level relates to a correspondent development level. Each step of the V-model requires different knowledge and skills. Thus the methods and personnel must fit within each phase characteristics, as in any other development methodology.

2.2 Software Quality

There are many definitions for quality of a product or service, such as *Fitness for use* (Juran and Godfrey 1999) and *Conformance with requirements* (Hoyer et al. 2001).

The same mindset expressed in the previous quotes should be applied in assessing quality in a software context. As such, quality in software is not limited to the absence of failures, but it includes the following factors (ISO 2011):

- functionality – required capabilities of a system. Through testing, it is possible to assess if the system was implemented according to the specifications;
- reliability – confidence in the ability of the system to operate during a specific period;
- usability – ease of learning and operating the system. Also, the attractiveness of use is key to excel at this factor;
- efficiency – balance between the tasks performed by the system and the required inputs for its operation;
- maintainability – code entropy, used to evaluate when it becomes cheaper and less risky to rewrite the code than to change it.

The previous characteristics should be prioritized to define what depth should be applied when testing each of these aspects (Schaefer 2014).

⁵ Environment – a testing environment is a controlled setup of software and hardware where the tests are executed to verify the developed software quality

2.2.1 Software Anomaly

To employ the term anomaly as a synonym for error, glitch, defect or bug deemphasizes any eventual distinction between these words. As a result, it is recommended for each of those entities to associate a particular definition to establish differences (IEEE 2010).

The current terminology used at Deloitte's premises is that a defect is an anomaly present in the software system. If it is an existing problem at production and thus not responsibility of Deloitte, then the terminology is incident.

2.2.2 Origin of Defects

There are many reasons associated with the insertion of defects into software. It is worth considering some common systematic causal analysis to tackle the underlying weaknesses (Kaner 2002).

When studying the origin of defects, five dimensions must be considered. Each one of these five dimensions is associated with a stage of software development life cycle and its most common defects (Kumaresh and Ramachandran 2012):

- requirements analysis – inaccurate requirements guarantee that expectation are unmet. As such, it is critical to identify defects related to requirements before their incorporation in the following design and implementation phases;
- design flaws – frequently projects fail due to inadequate design. Since a system cannot operate without proper infrastructure, a faulty design leads to severe defects later;
- defective coding – if an application contains coding errors, it means that the design was not correctly executed, thus producing unexpected behavior to its operation. Since this particular dimension will appear with considerable relevance, the main root causes for this type of defect are presented in Appendix A;
- delinquency in testing – improper testing promotes the escalation of defects into higher environments. As in the previous dimension, due to the importance of proper testing to the scope of the dissertation, the most common causes of defects in this dimension are listed in Appendix B;
- duration slippage –frequent changes in the business may lead software to become obsolete if the development does not keep up with the new context. However, the effort to keep updated requirements can in turn lead to a busy schedule favorable to the introduction of defects.

2.2.3 Relevance of Defects

From the developer's point of view, defect reports provide vital feedback to assess the quality of his work and where improvements can be deployed⁶ (Bettenburg et al. 2008). Also, as defects progress down the development path, fixing problems is considerably more expensive. Thus quality control must start at the requirements analysis and early developments, to avoid costly downstream fixes. (Boehm and Basili 2005).

Although some of the concepts for improving quality in manufacturing products can be applied to the software industry, a special approach is required since software is developed and not produced. In other words, concerning customer decoupling point, software is engineer-to-order and not make-to-order. Make-to-order manufacturers can use statistical quality control as the operations under analysis are repeatedly performed under the same conditions, while this type of control is impossible to software (Basili and Caldiera 1995).

⁶ Deploy in a software context is a procedure through which updates are implemented into a software system

Still, initiatives such as defect prevention can be adopted to increase the quality of the software product while reducing overall costs and required resources. By addressing the root causes of defects, preventive mechanisms may be established to reduce re-occurrences of similar defects later in the software life cycle or even in subsequent projects (Kumaresh and Baskaran 2010).

It is possible to improve the overall performance of the organization with effective mechanisms of communicating lessons learned and measuring tradeoffs between the investment in causal analysis and potential returns in quality and productivity (SEI 2010).

2.2.4 Costs of Finding and Fixing Software Defects

The relative cost to fix software defects follows the trend expressed in Figure 4.

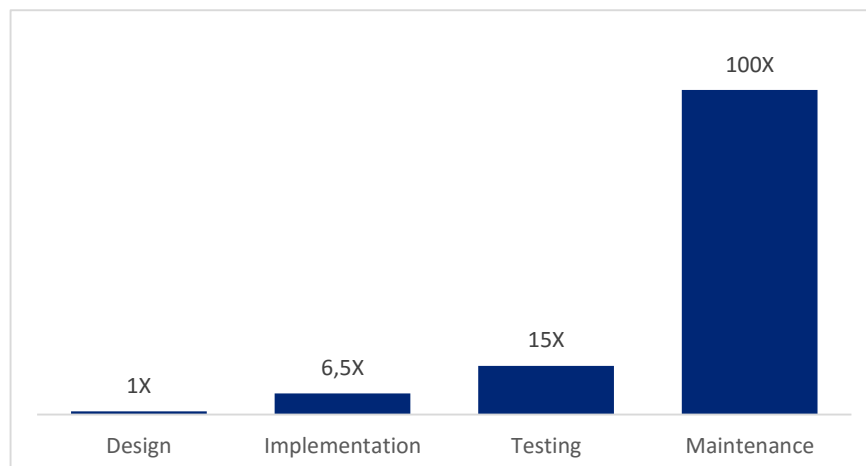


Figure 4 – Relative cost to fix software defects in each phase of its life cycle (Dawson et al. 2010)

The relevance in understanding the many processes in software development and assuring their quality is crucial, since the later a problem is found, the higher will be the cost to fix it.

2.3 Testing

Testing is the activity responsible for assuring the quality of a software application. In this section, a few considerations regarding the responsibilities of the tester and the risk-based approach practice are presented.

2.3.1 Responsibilities of the tester

Concerning the tester's job, his primary work product is his defect reports. However, a tester must not be encouraged to find as many defects as he can per se, but instead to get the most defects fixed. Moreover, the tester should be shaped so as to provide effective information for an informed business decision on the resolution of the defect. Once a tester detects a defect, it is also his responsibility to get it fixed (Kaner 2002).

2.3.2 Risk Based Approach

Often all other activities before test execution are delayed. This means testing has to be done under severe pressure. It is out of question to quit the job, nor to delay delivery or to test badly. The real answer is a prioritization strategy in order to do the best possible job with limited resources. (SCHAEFER 2006)

It is crucial to identify where the cost of failure is higher and where the probability of failure is higher to make good use of the available resources. If test execution reveals defects clumped in an area, it is highly likely that there are more defects to be identified in that very area as a result of development difficulties. Thus, at the end of the testing phase, it is important to generate more tests aimed at looking at the prone to defect areas. Moreover, planning contingencies and understanding the risk of the available alternatives is crucial (SCHAEFER 2006).

Five strategies are provided at the quoted source for mitigating the risk of delivering a bad product, namely:

- finding the right quality level: for example, performance metrics such as Return on Quality, assess where to place the quality level to achieve the best financial performance;
- prioritizing test execution by checking which functions are critical and which are not, where defects are more likely to occur and execute tests accordingly;
- making test execution cheaper: for example, automation of test execution is a potential cost reducer;
- defining entry criteria: demand a minimum level of test coverage at unit testing (tests performed by the developer);
- preventing the reoccurrence of the same defect through continuous improvement in the development process activities.

2.4 Defect Reporting and Classification

In this section, relevant references considering reporting good practices and classification rules are studied.

2.4.1 Reporting - Good Practices

When reporting defects, it is advisable to (Kaner 2002):

- explain how to reproduce the problem;
- limit a report to a single defect;
- justify why the described behavior is a defect.

Also, since the information on the defect is to be shared between different team members, there must be a common denominator of fields in the report containing useful information (Kaner 2002).

2.4.2 Classification of Software Defects

The standard IEEE Std 1044-2009 consists in a uniform approach to the classification of software anomalies. The purpose of this standard is to define a common language through which different people and organizations can communicate effectively concerning software defects, while establishing a common set of attributes that support data analysis techniques.

To set a standard framework for classification, the organization must define (IEEE 2010):

- 1) the objectives to be achieved by the classification;
- 2) the reference standard;
- 3) conflict management regarding classification decisions;
- 4) the start and end for the classification scope within the project or product life cycle;
- 5) the classification attributes - Table 2;
- 6) who is responsible for the classification attributes' assignment;
- 7) classification data infrastructure.

Table 2 – Defect attributes: adapted from (IEEE 2010)

Field	Short Description
Defect ID	Unique identifier
Description	Description of what the problem is
Status	Current state within defect life cycle
Asset	Software module or component containing the defect
Artifact	Specific software work product containing the defect
Version detected	Software version in which the defect was detected
Version corrected	Identification of the software version in which the defect was corrected
Priority	Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect
Severity	Highest failure impact that the defect could cause determined by the organization responsible for software engineering
Probability	Likelihood of recurring problems caused by this defect
Insertion activity	Activity during which the defect was introduced
Detection activity	Activity during which the defect was detected

Moreover, the benefits of having a standard way of classifying include (IEEE 2010):

- information on the most common type of errors during development and testing – valuable input for process improvement with, for example, causal analysis;
- better communication and exchange of information – since people frequently associate with the same word(s) different meanings, it is relevant to have a common understanding of the terminology employed.

2.4.3 Defect Rejection

Not every defect reported is necessarily a defect. As it can be seen in Appendix C, many reasons can be associated with defect rejection.

According to Zaineb and Manarvi (Zaineb and Manarvi 2011), the most problematic area causing this course of action, according to the source, is insufficient knowledge of the tester over the software being tested. Thus, instead of reviews and reworks, proper training should be provided to the testing teams to make the testing execution as simple as possible. Also, testing and development members should be part of the team gathering requirements. This way requirements are better understood and, more particularly, is created awareness to possible technical impediments for the solution to be delivered according to the expectations.

2.5 Microsoft Team Foundation Server

Since the client uses Team Foundation Server to manage requirements, tests, and developments, this section will describe the most important aspects of the tool.

2.5.1 General Information

Microsoft Team Foundation Server, from now on referred to as TFS, is a collaboration platform that offers project tracking, data collection, reporting, source control and work tracking tools. Therefore, it is a solution created to aid developers, project managers and test managers throughout the application life cycle (Microsoft 2010c).

In another perspective, TFS can be seen as a tool that supports the software development process by performing many processes that would be time-consuming while easing the communication between project participants due to the availability of project data (Microsoft 2010a).

This tool integration with SharePoint allows users to set a website easily with relevant data from TFS and automated reporting tools to provide online feedback on the progress of the application life cycle (Microsoft 2010b).

2.5.2 Test Management

Figure 5 describes a possible representation of TFS support on testing activities. The fields at the top represent the business requirements and system specifications to be placed under test, through test cases. To these elements, defects may be associated if there are deviations from the expected result when executing the test case.

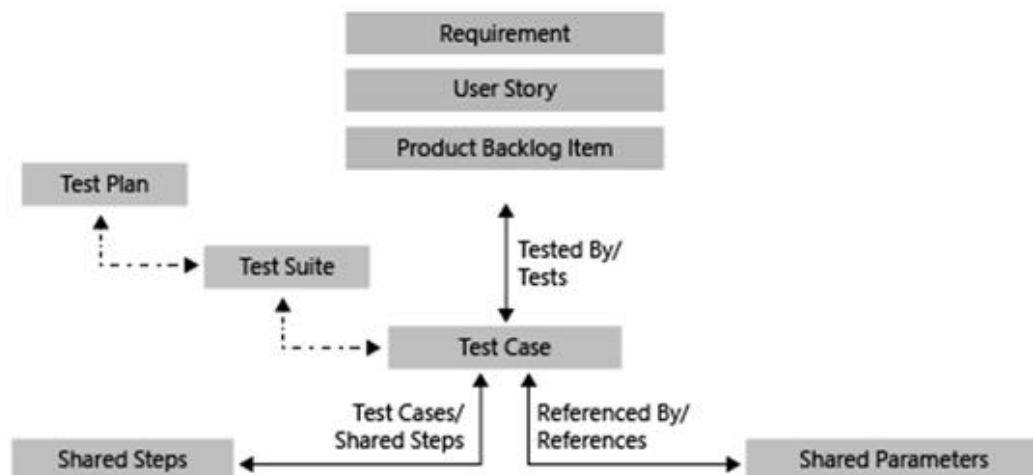


Figure 5 – Adapted from TFS test management (Microsoft 2012)

Test cases for the same requirement are organized in a test suite, which is in turn convened under a test plan. Test case with common sequences of steps or parameters can be associated with shared steps and shared parameters, respectively (Microsoft 2012).

2.5.3 Work Items

The smallest unit of a task in TFS is called work item. These items are designed to assist the development teams in managing and keeping track of software defects and potential improvements. An alternative definition is that a work item is a database record that contains the definition, assignment, priority, and state of work. Also, work items can be linked to each other when there are dependencies.

A work item usual life cycle, following the CMMI template (Microsoft 2015b):

1. PROPOSED - a work item has been created/found;
2. ACTIVE - it has been recognized as a piece of work that must be completed;
3. RESOLVED - the work item has been resolved;
4. CLOSED - the work item has been closed and is completed.

Figure 6 presents a typical life cycle for the work item defect.

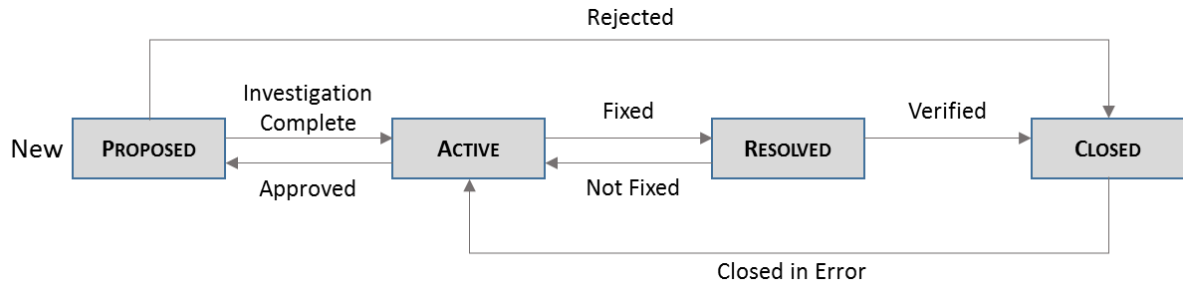


Figure 6 – General life cycle for a defect

Defect

At any time during the software development, it is possible to report a defect. When creating a defect in TFS, there is a set of fields that must be filled to make the defect acknowledgeable to others. In Table 3 is listed the fields from Table 2 that are present in this tool, with the addition of ASSIGNED TO, which contains the next responsible element along the life cycle (Microsoft 2015d).

Table 3 – Defect description: relevant fields

Field	Field
DEFECT ID	DESCRIPTION
STATE (STATUS)	ENVIRONMENT (VERSION DETECTED)
PRIORITY	SEVERITY
HOW FOUND (DETECTION ACTIVITY)	DESCRIPTION
AREA	ASSIGNED TO

Test case

In the core of the test case lies a group of steps containing what actions must be executed and what the expected results are for them. This way, the test cases are made available to the team elements with relevant information enabling their execution by any member of the project.

At the same time, information related to their state, for example, ACTIVE, IN PROGRESS, PASSED or FAILED, is shared online to keep track of their progress. Also, to a test case may be assigned a PRIORITY, according to its relevance for the project (Microsoft 2015c).

2.5.4 Source Control - Branching and Merging

Source control provides automated track of changes in code and documents. It makes sure at the same time that changes are not lost. The common usage follows the pattern that if a user needs to change a file, he searches for the last version, edits it and uploads it back into the repository with a comment so that every other participant can see who has changed what and when (Microsoft 2015a).

In particular, Branching and Merging allows individuals to work in parallel by allowing work to be broken down into smaller pieces which are to be reassembled later on. A summarized diagram of the process is presented in Appendix D.

2.6 IT Management Standards

A considerable amount of effort is required for organizations to change the work habits of their staff and the risk of the process is considerably high. However, successful companies take advantage of IT to drive their stakeholders' value up and manage the associated risks effectively (Chen, Chern, and Chen 2011).

2.6.1 Process Relevance in Quality

There are three dimensions worth considering for organizations to develop and maintain the quality of their services and products. These dimensions, presented in Figure 7, are brought up together through processes, which are the vehicles for the organization's workforce to *meet business objectives by helping them to work smarter, not harder, and with improved consistency* (SEI 2010).

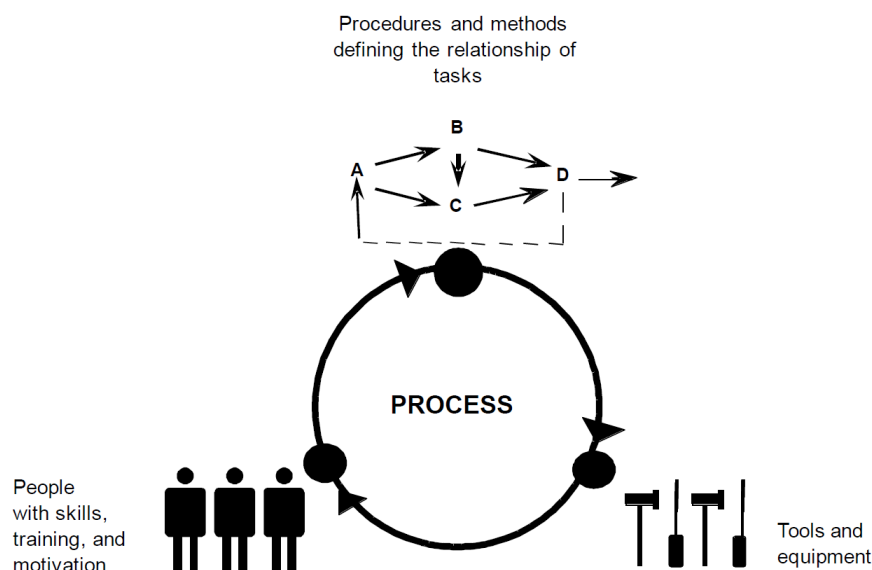


Figure 7 – Critical dimensions for quality improvement (SEI 2010)

Awareness of process improvement relevance in efficiency and effectiveness can be traced to the 30's, with Shewhart and the principles of statistical quality control, following the premise of *the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it* (SEI 2007).

2.6.2 Capability Maturity Model Integration (CMMI)

CMMI is a process improvement training that addresses capabilities gap in organizations. Effective management of organizational assets is critical to achieve business objectives, and CMMI provides a systemic approach to address the problems most organizations face (CMMI 2016).

Capability levels are well-defined evolutionary steps describing the organization's capability relative to a process area. Appendix E presents a table with the different capability levels from CMMI. Generic goal and generic practice descriptions translate how the process is regarding integration and consistency. The higher the integration level, the more capable is the organization to provide valuable services with better quality, lower cost and faster. A capability level for a process area is achieved when all of the generic goals are satisfied up to that level (SEI 2010).

Two of these Capability Levels, namely CL-2 MANAGED and CL-3 DEFINED are relevant in the context of the present dissertation, since the actual and desired state are established considering these two levels.

So, relative to the first one a managed process *is a performed process that is planned and executed in accordance with policy; employs skilled people having adequate resources to produce controlled outputs; involves relevant stakeholders; is monitored, controlled, and reviewed; and is evaluated for adherence to its process description* (SEI 2010). The objectives of the process are established by the organization. Commitments between the staff and the stakeholders are reviewed and controlled to guarantee that the product or service satisfy their specific needs. In Appendix F, detail of each specific generic practice is presented for CL-2 MANAGED.

Concerning the second one, a defined process *is a managed process that is tailored from the organization's set of standard processes according to the organization's tailoring guidelines; has a maintained process description; and contributes process related experiences to the organizational process assets* (SEI 2010). The organizations' set of defined processes is described in more detail than a managed process and as a consequence improvement information is easier to understand, analyze, and use. Also, the interrelationships between process activities are clearer in defined processes. In Appendix G, detail of each specific generic practice is presented for CL-3 DEFINED.

3 Problem Analysis

Being the effectiveness of a solution critical to its success, in the following chapter gaps in the defect classification and reporting processes will be analyzed and structured.

Firstly, the requirement methodology and testing phases are presented to set a particular context for the initiatives under study. Secondly, two As Is models are presented, one based in documentation and the other in observation. Based in this two models, the gaps present in the processes of handling and classifying defects are identified.

In the end, the diverse aspects of the problem to be tackled will be summarized to acknowledge how the solution can serve best the Deloitte's teams and ultimately the customer's needs.

3.1 Requirement Methodology

To provide context for testing in the requirement implementation process, part of the process to deliver them into production is presented in Figure 8.

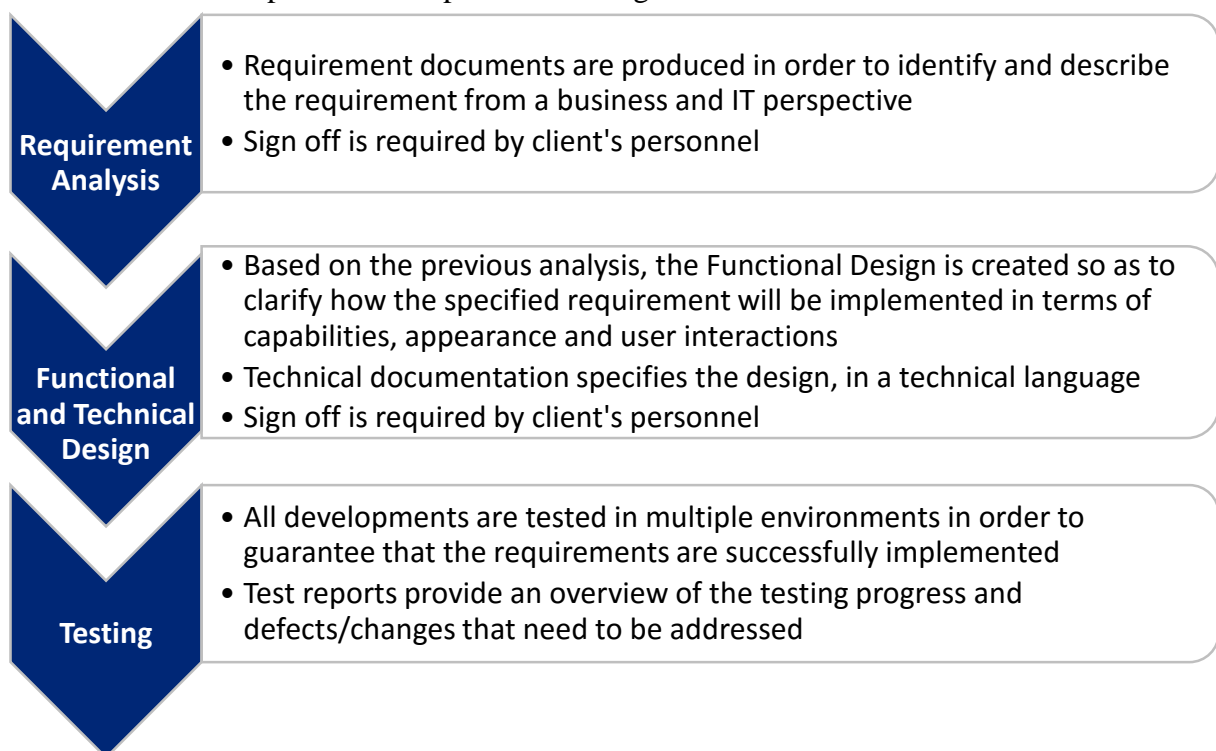


Figure 8 – Requirement delivery overview: functional perspective

3.2 Testing

There are several testing phases as presented in Figure 9, each one with a specific scope, an expected output, and a responsible team. There are established standard reporting and meeting procedures for all the test phases.

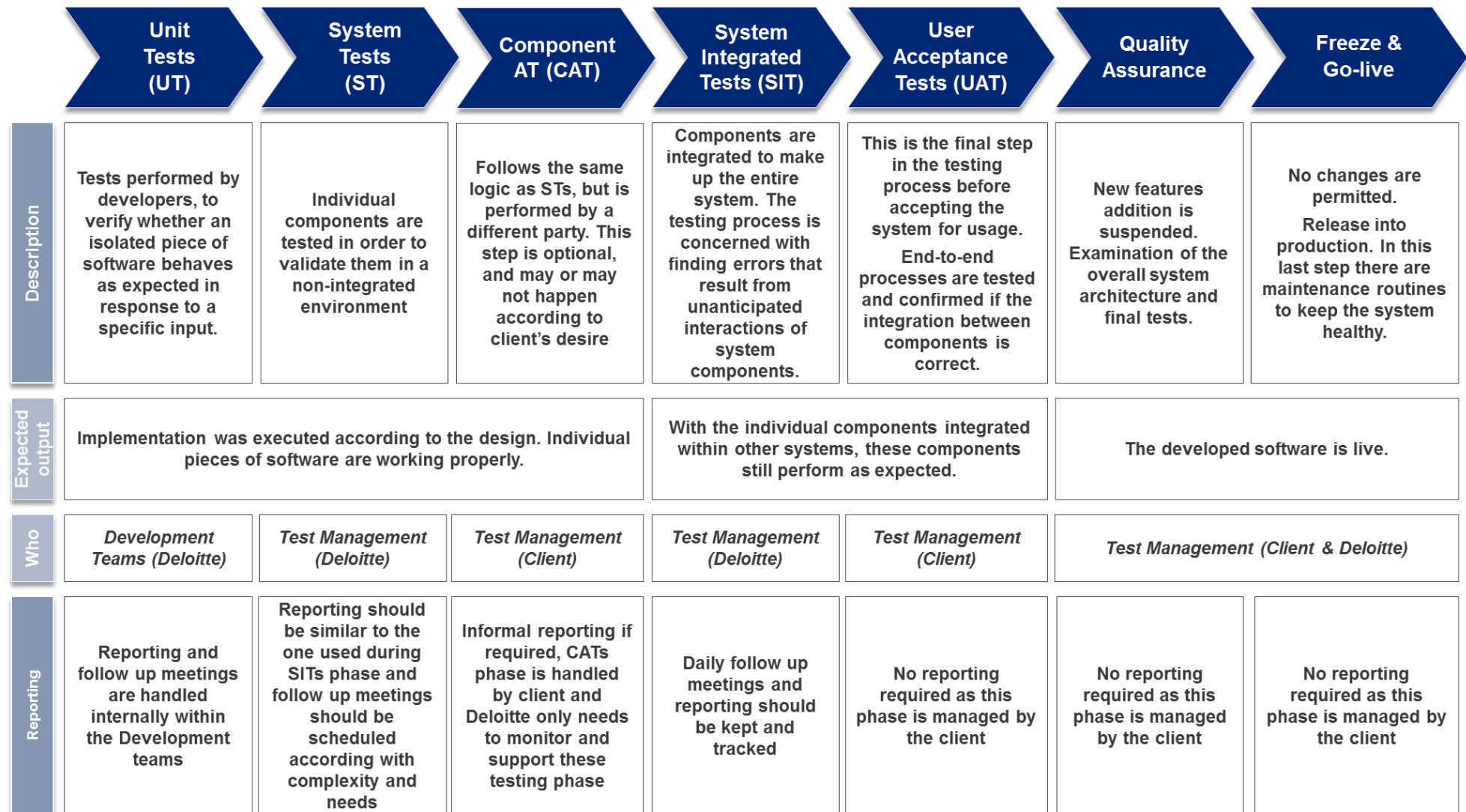


Figure 9 – Testing methodology overview: Deloitte's perspective

Considering the architecture of the client's information systems, two levels can be defined. The first one being limited to the core system and the second one embodying every individual system, such as front-ends and external entities, in an integrated environment.

To understand the relation between the previous testing phases and those two levels, Figure 10 illustrates at which level each phase takes place.

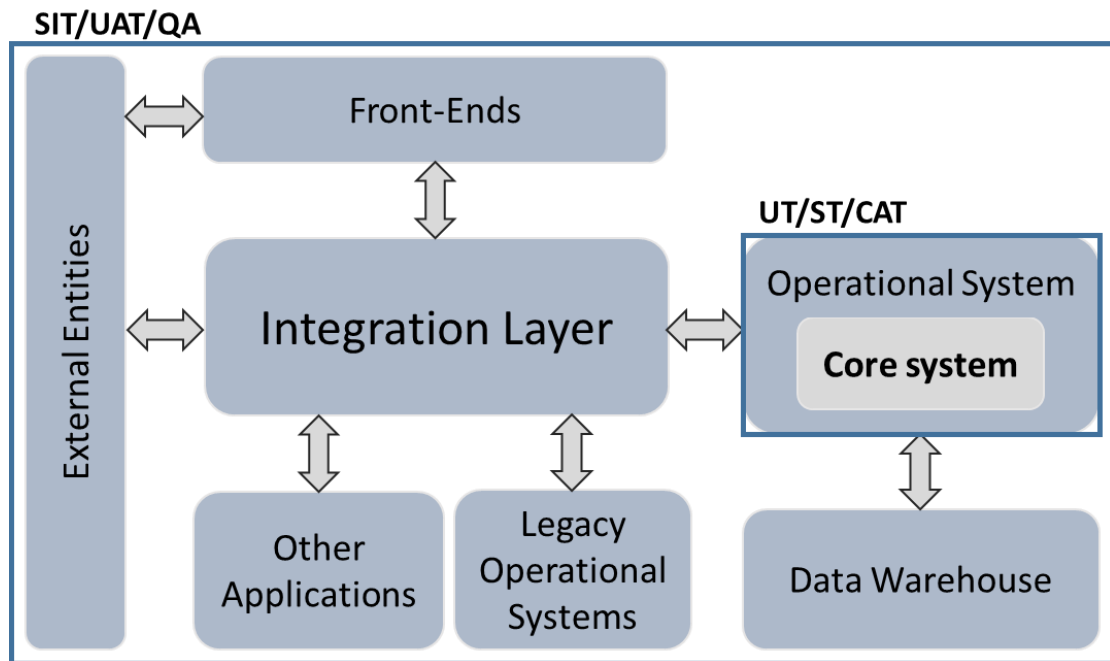


Figure 10 – Information system architecture: test phase scope

3.3 AS IS - Documented

In this section, the information documented by Deloitte and the client concerning testing and defect reporting procedures is presented.

3.3.1 Deloitte Documented Practices

In this section, the documented Deloitte's practices for this specific project are presented.

Defect Life Cycle

At the beginning of each test phase a defect manager should be clearly identified. This defect manager consists in the one responsible for the deliveries of the implementation team. Defects must be created with the status **PROPOSED**, which is pre-populated by default, and should be assigned to the accountable defect manager.

Active defects are defects that are still under analysis or that are being fixed. Everyone involved in the testing and fixing process must keep continuous tracking of the information in the TFS, to identify and promptly manage every new defect. When the development teams receive a **PROPOSED** defect, they must change the state to **ACTIVE** and decide whether it should be re-assigned to the tester as **AS DESIGNED** or proceed to be resolved.

After a defect is fixed, it must be assigned to the defect manager and set to **RESOLVED**. Only after deploying it, the fixed defects should be re-assigned to the tester who reported it for retesting.

If a defect is not fixed then it should be re-set to ACTIVE. Only after successfully deployed and retested it can be set to CLOSED and then it should be tracked as ‘Defect resolved and deployed. Re-testing passed and defect closed as expected’. If a member of the customer team opened a defect, then it must be closed by the client, namely by the person that opened the defect.

The normal flow follows the sequence: PROPOSED – ACTIVE – RESOLVED - CLOSED. However, there are other options for state transitions. It is possible to change from the state PROPOSED to RESOLVED when the defect is DUPLICATED or REJECTED. It is not recommended to set a defect to CLOSED before going through the state RESOLVED. A defect set to CLOSED may be re-opened to state ACTIVE if necessary. However, once a defect is created its state should never revert to PROPOSED.

Figure 11 presents a detailed defect life cycle. Note that the representation is from Deloitte’s point of view and that some of its characteristics will be better analyzed in section 3.3 – Defect creation and handling.

After a defect is closed, information on its resolution must be filled in by the technical team and then approved by the tester. This nomenclature can be observed in Table 4.

Table 4 – RESOLVED REASON nomenclature

RESOLVED REASON	Short Description
AS DESIGNED	The behavior is the one expected according to the design
CANNOT REPRODUCE	The technical team has been unsuccessful at replicating the issue
DUPLICATE	The defect has been already reported
FIXED	The defect was successfully solved
REJECTED	Defects that have been wrongly created

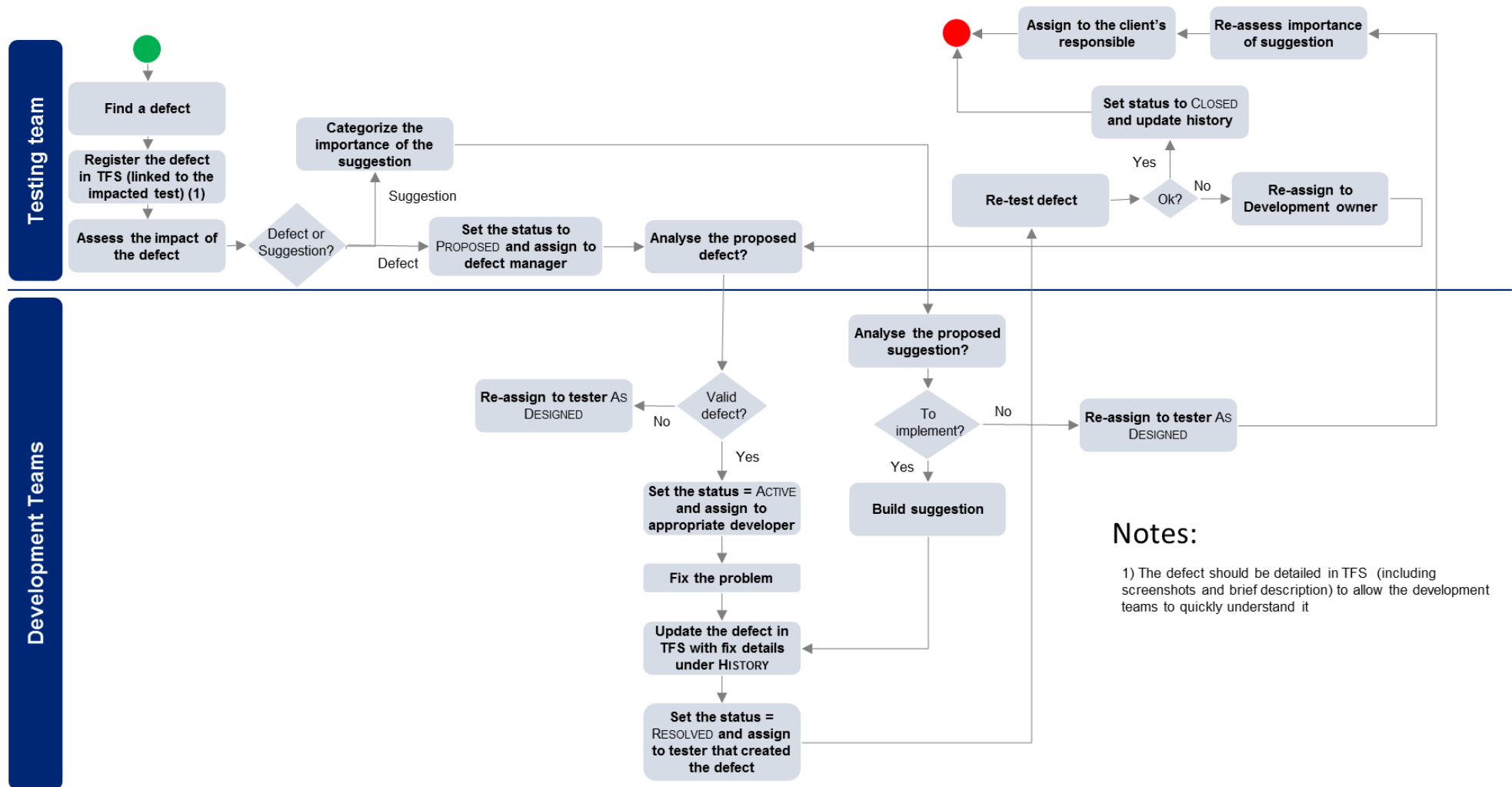


Figure 11 – Generic defect life cycle

Knowledge Share and General Practices

To support knowledge transfer initiatives directly developed to this specific client, Deloitte provides an eRoom with How To and Lessons Learned documents.

As best practice, this eRoom should also be used internally by Deloitte to support work in progress, such as testing documents under review. All documents to be shared with the client should be then passed and uploaded to the customer's SharePoint.

According to Deloitte's standards, when a defect is found and reported at least one test case should be linked to the defect and, in the case more than one test is affected by the same defect, all the affected tests must be linked to the same defect.

If a defect was associated to a test, then retesting must always happen to assess if the behavior reported in the defect has been fixed. The state of a defect may only be set from RESOLVED to CLOSED after proper retesting of the associated test case(s).

Deloitte clearly establishes that if there is any problem with a step of a test, such as inconsistency or omission, it must be reported to the test designer for him to review it and update the script.

Defect Creation and Handling

There are guidelines to follow when opening a defect, but since most of them are covered in sections 2.4.2 and the remaining ones have a low degree of relevance to the defect handling processes, for this dimension only SEVERITY and HISTORY will be analyzed in the present section.

Table 5 – SEVERITY field criteria

SEVERITY	Short Description
1 - CRITICAL	Defects that impact critical business areas. These defects are show-stoppers as further testing may be blocked until the issue is fixed
2 - HIGH	Defects that result on significant problems from a business perspective and impact system functional areas. These defects can lead to constraints for testing and/ or impact other system areas and should be fixed as soon as possible
3 - MEDIUM	Defects that represent minor problems on a single screen or on single system functionality for which there is a workaround
4 - LOW	The defect has minimum or no impact on the system functionality
5 - SUGGESTION	When the problem found is out of scope, not covered by the functional design, represents a design error that needs amendment and/ or is related to system enhancements found during testing phases.

Note that if the SEVERITY level is 5 - SUGGESTION, the defect should be addressed to as suggestion and not defect. Work item fields, such as RESOLVED REASON, are still to be filled in this case.

THE HISTORY field is a history log of the defect management including detailed information about changes in the defects states changes. For instance, when a defect is closed a justification must be registered in the HISTORY concerning the attribution of the RESOLVED REASON. In case it was FIXED, this comment must contain details on the defect resolution.

It is possible that once the technical team has analyzed a defect it is deemed either AS DESIGNED, suggestion or AM, the latter meaning Application Maintenance, which handles defects present in the production environment. The right procedure for each case might be different depending on whether the defect was opened by a Deloitte team member or by a client team member.

Defects deemed AS DESIGNED should have a clear reason for this classification. When a defect opened by client team members have received this classification, the reason for this should be clear and in this case the right procedure is to reassign the defect to the person that opened it, which will then decide whether to close the defect or turn it into a suggestion.

When a defect is deemed AM, client testers should be e-mailed to make sure that the behavior described is, in fact, observable in production, since Deloitte's testers do not have permission to access production environments. The e-mail should be accompanied by prints. If a test phase ends and the defect turns out to be an observable behavior in production, the defect should be closed and the AM team will, in turn, create an incident. An incident will be opened for every defect in the described situation and it is up to the client to decide on the consequent course of action.

If a suggestion comes from a client's tester, by the end of the process the defect will be reassigned to a customer's responsible, who will analyze whether to turn the suggestion into a requirement or finish this process.

When a defect opened by a Deloitte team member is deemed AS DESIGNED, DUPLICATE, REJECTED or CANNOT REPRODUCE, it should not be closed without verifying that the classification matches the observable behavior in the testing environment.

Acceptance Criteria

A PRIORITY is assigned to a test case and a SEVERITY is assigned to defects. For the following test phase to take place, the acceptance criteria for the previous test phase must be met.

The acceptance criteria for test cases are the following

- 100% of PRIORITY 1 test cases have been successfully executed;
- 99%-80% of PRIORITY 2 test cases have been successfully executed;
- 79%-60% of PRIORITY 3 test cases have been successfully executed;
- 59%-00% of PRIORITY 4 test cases have been successfully executed.

The defect must have been corrected with the following result:

- 100% of SEVERITY 1 - CRITICAL defects are solved – fixed and retested;
- 100% of SEVERITY 2 - HIGH defects are solved – fixed and retested;
- maximum outstanding of SEVERITY 3 - MEDIUM defects are 0-10 errors;
- maximum outstanding of SEVERITY 4 - LOW defects are 0-20 errors;
- maximum outstanding of SEVERITY 5 - SUGGESTION defects are 0-30 errors.

Test cases that have not been executed and defects that have not been solved in the respective testing phase progress to the next one on Deloitte's responsibility.

Preparing Test Cases

Deloitte's teams are responsible for the phases UT, ST, and SIT, and must prepare test cases for these phases. In UT, tests are performed by the developer himself. In these phases, TFS is used to handle defects and the responsibility for designing and executing test cases belongs to the test management.

Concerning writing tests scripts, Deloitte makes reference to the differences and similarities between ST and SIT phases and thus what should be taken into account in the writing process.

At ST, since the objective is to validate several independent components in a non-integrated environment, if the requirement under test also takes place in an external application to the core system, integration may be simulated to assure that the response is correct. While at SIT, as the objective is to test components that have been integrated to make up the entire system, test cases should take into consideration the various applications/tools involved when performing specific actions and should direct users to the documents and steps necessary to perform them.

Also, test cases from ST should be copied and adapted to SIT whenever possible.

Moreover, it is recommended to:

- make use of shared steps when writing tests with steps in common;
- keep test simple and precise, but complete and with sufficient detail – take into account what another person will need to execute the test;
- improve the scripts with shortcuts learned while performing tests, while making sure the other team members are aware of these findings;
- assure that the information written in a test case matches as closely as possible to the information made available in the functional design documents;
- use test case extractor to access previously executed tests that should be executed again, in a regression logic.

3.3.2 Client Documented Practices

The client has also documentation to support TFS related activities, but since this part of the analysis will be set mostly within Deloitte own resources, only a brief overview of relevant aspects on the client side will be analyzed here.

The client provides a user guide for TFS 2010, as this is the tool used in maintaining and handling requirements, test cases and developments during the application life cycle. It is supposed to be used as a reference for how to do a specific task in TFS.

In this document, most of the chapters relate to the guidelines in Microsoft's bibliography on how to use this tool. However, there is little to be added since the relevant aspects are already covered in section 2.5.

Also, the client promotes some workshops and meetings to understand how to improve the performance of the activities that maintain and deliver upgrades to the IT systems. The goal of these activities is to increase operational efficiency, speed, and agility, while mitigating risks, and improve communication and collaboration within and between teams. The top issues discovered in those workshops and meetings are enumerated below:

- communication between and within teams being too ad hoc;
- unclear roles & responsibilities;
- lack of visibility over end-to-end processes;
- misalignment between team leaders.

3.4 AS IS - Actual Practices

In this section, the actual practices are characterized through observation within the teams that deliver the system upgrades, historic analysis of previous defects and a survey conducted to the Deloitte's professionals working for this project.

3.4.1 Observation

Observation takes place in the second quarter of 2016 and is limited of functional teams under ST and SIT phases.

Testing

Tests are either performed manually or through automated functional tools, being the latter exclusive to regression tests which are meant to verify if new developments have not unintentionally induced alterations in unrelated features.

Concerning their design, test cases are built on a risk-based approach. The updates to the system cannot be tested in full, so, considering the available resources and the risk associated with not testing a particular part, it is decided a scope for the testing to cover. This trade-off is of the client's knowledge since, by principle, Deloitte team works closely with the customer's test management team to gain commitment and engagement from the customer side, while mitigating the risks associated with the risk-based approach.

The PRIORITY in the test case is filled with 1, the highest level, if the test is related with the developments that took place in the initiative under test and with 2 if it is a regression test. About the Shared Steps feature incorporated into TFS, it is not usual to take advantage of this feature.

Even though requirements are placed upon SharePoint with an associated degree of PRIORITY, this information is not taken into account when defining testing scope under the risk-based approach. If assumed that to a more urgent requirement is also more important to assure its correct implementation, then PRIORITY should be taken into account when defining the scope of the tests.

Furthermore, it is normal at the start of SIT phase for the integration layer⁷ to be unavailable. Added to the fact that the test scripts from ST are reused with added steps to cover associated changes outside the core system, a considerable part of this phase is a repetition of ST, as the same test is often re-executed. Also, due to this unavailability, defects are reported to reflect impediments on testing progress, even though it is not a defect introduced with the new requirements implementation but a problem at the integration layer.

In many high-level follow-up meetings important decisions are taken that affect the course of tests' execution, but it is not usual for the team to participate in them, except the team leader responsible for the deliveries. As such, relevant information is not reaching the team elements that execute the work thus promoting the occurrence of misalignment problems.

Besides, it was observed that testers do not usually enroll in the design phase and may only be added to the team after the workshops and meetings with the client. As such, the comprehension on the scope of the developments is mostly limited to their experience and to the availability of relevant information. In FD, TD, and SharePoint is made available a considerable amount of information on the developments, but it is not guaranteed that it covers every aspect to take into consideration.

Finally, test designer is not a defined role in the actual team's structure so, when changing test scripts, the initiative leader should be the one approving any alterations.

⁷ Integration layer – The communication between different systems composing the IT infrastructure of the customer is enabled through integration services

Defect Reporting

When reporting a defect, the TFS fields to be contemplated are listed Table 3.

To the SEVERITY is commonly assigned the value 3 – MEDIUM by Deloitte's teams and only in very particular cases this classification will escalate to higher levels. For example, if it is not possible to use a module of the system, the severity attributed in this case will be 1 – CRITICAL. Although SEVERITY is useful to understand how the environment under test and the developments are performing, this field does not hold a significant impact on the teams' workflow since any defect must be fixed despite its SEVERITY.

Although the acceptance criterion is deemed as a baseline to the testing phases, it is not impossible for its restraint to be overruled. In particular situations, it is possible to have two different testing phases active. In the case this situation occurs during SIT to UAT, the client is informed of what tests have not been covered in SIT and what has already been executed, as well as the defects opened that still have not been closed.

When is assigned RESOLVED REASON FIXED to a defect, details on the fix are not being included. Since defects are most of the time related to a specific test case, sometimes these objects are used not to address a single issue, but multiple ones that appear during the same test execution.

It is possible that a design document contains incorrect or absent information, inducing in error the tester in error. Since each requirement is often transversal to more than one document, which, in turn, are on the responsibility of different people, it is possible for the implementation to be correctly executed despite mistakes at design documents. Between business requirements, design and implementation gaps may be introduced, and thus a critical analysis must be performed when comparing design with developments.

3.4.2 Defect Historic Analysis

In this section, a part of the TFS historic will be studied to understand the actual practices in the past. This analysis will not only provide quantitative data but it will also provide a much clearer picture of existing gaps within functional and technical teams.

Comparison Between Releases

Since early 2013 to late 2015, about 12.000 test cases were executed and 6.400 defects reported over four different releases. Appendix H is dedicated entirely to the comparison between releases.

As it can be observed in Appendix H, although some differences can be noticed between the releases, because of the unavailability or dispersion of information on each release, it becomes difficult to understand the reasons for these differences and take lessons for the future releases. Thus, the historic will be analyzed as a whole.

Aggregated Analysis - Overview

Appendix I presents the results of the historic analysis. It was done by selecting the fields from TFS that contained relevant information on the actual quality of the defect classification and that could suggest eventual gaps in the process.

Within the defects analyzed, it is pertinent to note that:

- over 1/3 are rated as 2 - HIGH and 1 - CRITICAL under SEVERITY;
- 3/4 end without an associated ROOT CAUSE;
- 1/2 of the suggestions end as REJECTED;
- the most common root cause is CODING ERROR;
- 3/4 are found during SIT, UAT, and ST;
- 1/3 end up with a different RESOLVED REASON than FIXED, meaning that probably they were not defects.

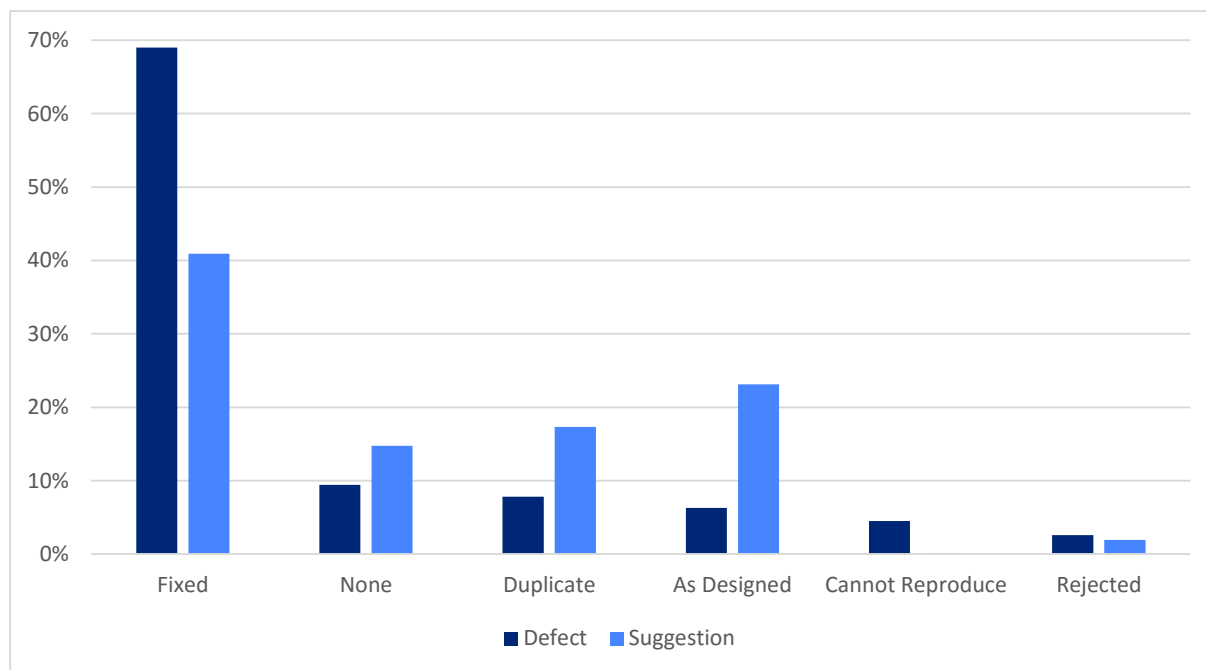


Figure 12 – A third of the defects reported end up not FIXED

Also, a quarter of the defects analyzed does not follow the defect life cycle defined in section 3.3 – Defect Life Cycle, either by not following the recommended transitions explained in that section or because the transitions occur at an inconsistent date.

Moreover, a quarter of the defect reported take more than two months to be set to CLOSED, with an average of fifty days. This reality is due to, for example, several different issues being reported in the same defect over the course of executing a test case. Concerning suggestions, the average increases to two hundred days.

Since the effort to prepare the data on the closure time of the defects under study would involve understanding specificities of each one that are not directly accounted at the work item defect fields, further analysis on closure time was abandoned.

Severity

During SIT, 40% of the reported defects are associated with SEVERITY 1 – CRITICAL or 2 – HIGH. From observation, a possible cause for this reality is unavailability at the service layer, since it has a significant impact over the system functions and testing progression.

Root Cause

Most of the defect in the historic of the studied releases were left without a root cause for their creation.

If to the ROOT CAUSE is joined RESOLVED REASON value, it is observable that CODING ERROR is by far the most common. It occurs in defects that not only end up as FIXED but also with the remaining resolved reasons. If CODING ERROR were the cause for a determined behavior to be reported, then it would make sense to understand if the problem reported appeared due to the developments under the test or if it was a problem in production.

Concerning suggestions, to the majority is associated DESIGN ERROR as ROOT CAUSE, which is understandable due to its very nature. However, the historic revealed some attributions of CODING ERROR and considering the definition of suggestion it does not make sense to perform such attribution. In addition, there are no guidelines to the attribution of the ROOT CAUSE, and thus it is susceptible to the opinion of the technical element handling the defect.

Environment

Concerning ENVIRONMENT field, it was observed that three-quarters of the defects reported occurred during ST, SIT, and UAT. Since these phases have the most intense testing effort and have many similarities between them, specific aspects concerning them will be analyzed further on.

Resolved Reason – Causal Analysis

Since the history of a defect is rich with information on the attribution of RESOLVED REASON and there is a considerable amount of false defect reports, the improvement suggestions will focus particularly on this dimension of the classification process.

One-third represents over two thousand defects with questionable nature over less than three years. It is possible to observe that many times the reasons for the classification are related to gaps in the classification practice by studying the HISTORY of these questionable defects.

A randomly selected sample of ninety defects with RESOLVED REASON AS DESIGNED, CANNOT REPRODUCE, DUPLICATE and REJECTED was studied. Within this sample, there are twenty-one defects without justification of the close reason and other six defects which are written in the client's native language, Danish. The sample size followed a stopping criterion of not finding any new root cause for this causal analysis within five new defects.

After the analysis of the sample, the reasons for the classification presented in the history were compiled into a few root causes for their attribution, associated with their number of occurrences level and a short description, in Table 6. Note that the number of occurrences levels are presented in Table 7.

Table 6 – Root causes per RESOLVED REASON (Historic analysis)

	Short Description	Levels			
		AS DESIGNED	CANNOT REPRODUCE	DUPLICATED	REJECTED
Incorrect testing procedure	The test was not properly executed	4			1
Flawed functional design	Errors in functional design induced in error the defect reporter	1			
Technical limitations	Due to technical impediments, the conditions were not appropriate for testing	2			
New requirement / valid suggestion	The reported issue has been transformed into a new requirement / suggestion	2		2	
Available workaround	Since there is possible workaround, this is not an issue	1			
Different testing conditions	The conditions were not appropriate for testing		3		
Cannot replicate	Following the steps, the reported symptoms are not found		2		
Replicated defect report	Related defect opened for the environment in test			4	
‘As Designed’	Expected behavior			1	2
Waiting for deployment	A fix will only be available after the deployment			1	
Technical Limitations	Due to technical impediments, the conditions were not appropriate for testing			1	1
Deferred defect	Defect identified in a previous released but postponed to a future one			1	

As it can be seen, incorrect testing is a very frequent justification used to close a defect as AS DESIGNED.

Table 7 – Number of occurrences levels

Number of occurrences interval	Level
[1-3]	1
]3-5]	2
]5-8]	3
>9	4

Concerning the distribution of RESOLVED REASONS per ENVIRONMENT, it differs significantly in the three phases ST, SIT, and UAT, as is presented in Appendix I on slide 7. The most important aspects to retain are:

- ST has double of the percentage of AS DESIGNED when compared to SIT;
- SIT reveals nearly twice the percentage of DUPLICATE when compared to ST.

Also, note that the UAT phase has:

- the least amount of FIXED defects compared to other testing phases;
- the highest deviation among releases;
- the highest percentage of the AS DESIGNED and DUPLICATE – over half more than the maximum between ST and SIT.

In particular, at UAT DUPLICATE is often being used many times to address situations where a new requirement covering the reported issue has been created. Table 8 shows three entries of 'As Designed' to justify the attribution of RESOLVED REASON DUPLICATE, which occurred at UAT phase.

3.4.3 Survey

A survey was conducted to complement the As Is – Actual Practices. In this survey, the results of the historic analysis were presented and follow up questions were asked so as to gather different opinions from other elements participating in this project.

The survey was sent by e-mail to 130 Deloitte's professionals associated to the transformation program, having obtained 35 answers. Appendix J presents the analysis of the survey answers. Among the 130 elements, the distribution of roles – analyst, consultant, senior consultant, and manager - was equitable.

Concerning the characterization of the sample, the majority of the responders:

- are at a career level of Senior Consultant or equivalent and Manager – in other words, belong to the management layer of this consultancy firm;
- have over one year of experience in this particular project;
- perform/performed in technical roles, implementing requirements and handling defects.

Resolved Reason – Causal Analysis

In Table 8, the main root causes chosen for each RESOLVED REASON, except FIXED, are presented. Note that for these questions multiple options could be selected for each item.

Table 8 – Root causes per RESOLVED REASON (Survey)

RESOLVED REASON	Root causes	Level
DUPLICATE	- Not checking for similar defects opened under the same testing phase	3
	- Lack of communication	3
AS DESIGNED	- Due to technical impediments, the conditions were not appropriate for testing	3
	- Incorrect testing procedure	3
	- Ambiguous/flawed design	3
CANNOT REPRODUCE	- Unclear/absent reproduction steps	3
	- No available environments / different testing conditions	3
	- Description without the relevant data	2
REJECTED	- ‘As Designed’	2
	- Core system limitations	2
WITH NO ASSOCIATED REASON	- Doubt in classifying the defect	3
	- Lack of knowledge on the TFS usage guidelines	2

Table 9 – Percentage levels

Percentage Interval	Level
0% - 25%	1
25% - 50%	2
50% - 75%	3
75% - 100%	4

On Rejected it was observed under section 3.4.2, particularly in Table 6, that its attribution was being justified with the comment ‘As Designed’. In the survey an item was affected to this subject, asking the respondent if, in their opinion, AS DESIGNED and REJECTED were being used to address the same nature of causes. Almost three quarters answered Yes to this question. However, the remaining responders, who answered No, provided valuable insights for relevant distinctions. An example for an eventual distinction is that AS DESIGNED is used when there is a justifiable reason present in the functional design to contest the claim of a certain behavior being a defect. On the other hand, REJECTED is applied when there is no available design to contest the claim.

This particular case demonstrates how different people can interpret the same concept in many ways since there are no guidelines normalizing the classification of defects.

In the following two sections, *Technical Team* and *Functional Team*⁸, an analysis on the questions exclusive to members of technical teams that handled defects and members of functional teams that performed tests is presented.

Technical Team

The answers revealed that FIXED defects are considered by the respondents to be the ones with the most work involved, having around 4 in an ascending scale of work from 1 to 5. However, to the remaining RESOLVED REASONS, the classification of work did not lag far behind, being close to 3.

Moreover, the value associated with this kind of defects ‘that are not defects’ was deemed rather low and the inconvenience caused by them considerably high.

Asked if the functional team did a proper work before opening defects, the responses tended to the downside. Some answers associated with the No response were:

- ‘Before opening a defect, we need to ensure we know the expected behavior, not only based on Functional Documentation;
- ‘If the proper care was made the As Designed defects would not be opened. Sometimes all that is needed is a clarification with the technical team. Most of the times this is not requested’;
- ‘They do not understand the whole flow of what they are testing and do not study enough the application they are testing. Therefore, they do not have the knowledge to make a proper triage of these situations’.

Functional Team

According to the responses obtained for the set of questions directed to testers, the following considerations can be taken:

- It is not frequent to have doubts considering if a certain behavior is a defect or not;
- The time spent investigating a potential defect ranges from fifteen minutes to an hour, being the effectiveness of the research usually high;
- The main obstacles to decide if a certain behavior should be associated with a defect are the ambiguous/incomplete documentation and lack of knowledge on the application being tested.

Also, when writing test cases, analyzing the workshops with the client, looking up related changes in FDs and TDs and checking the requirement entry at SharePoint was considered to be essential.

To All the Inquired

The main obstacles for an effective and efficient interaction between functional and technical teams, according to the responses obtained, are:

- heavy workload;
- lack of communication rules and procedures;
- delay between implementation and testing.

⁸ Functional Team – refers to the elements that elaborate the functional design documents and execute tests to guarantee the quality of the developments

Moreover, improvement suggestions often contained the following topics:

- standardization of classification process;
- review of test cases with technical team;
- workshops performed by the technical team to explain how the requirements have been implemented and what the pretended new behavior is;
- filtering before defects reach technical teams;
- more useful functional design documents with better structure and richer information.

3.5 Summary

A brief summary of the analysis performed in the present chapter is illustrated in Table 10 to structure the areas of improvement to tackle in the chapter 4, Roadmap for Improvement.

Table 10 – List of problems identified

Areas of improvement	Problems
Communication	<ul style="list-style-type: none"> - misalignment between technical and functional teams - high-level reports and follow-up meetings decision are not shared with the elements performing the work
Classification practices	<ul style="list-style-type: none"> - different levels of knowledge and comprehension on defect handling practices - absence of guidelines to cover all the relevant aspects of classification - ad-hoc classification – lessons learned and knowledge transfer not being effective in a practical context
Testing methodology	<ul style="list-style-type: none"> - repeated testing - no standard prioritization practices
Roles and Responsibilities	<ul style="list-style-type: none"> - unclear roles and responsibilities within and between teams - no sense of an integrated environment and shared responsibility, where a team's work impacts the other

4 Roadmap for Improvement

To achieve the proposed objectives, a roadmap is presented in Figure 13 structuring how to deliver the new model. For this case, a template model chosen, namely the IDEAL, which stands for Initiating, Diagnosing, Establishing, Acting and Learning, from Carnegie Mellon Software Institute. To note, the scope of this dissertation ends at Establishing phase, specifically after some considerations concerning sub phase Develop Approach.

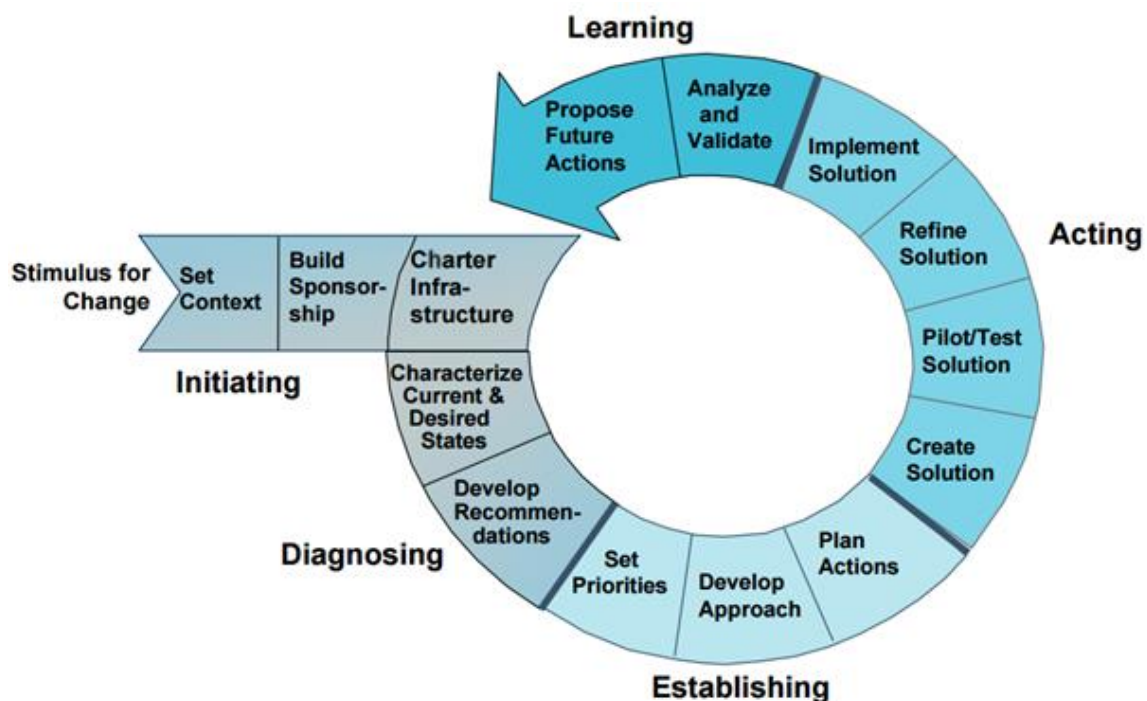


Figure 13 – Software process improvement framework (SEI 2009)

4.1 Initiating Phase

The conditions required for a successful implementation are studied in this section, namely the business reasons that justify the effort, the relevant managers support and the organizational infrastructure to deal with the process improvement details.

4.1.1 Sub phase 1 - Stimulus for Change

Both Deloitte and the client are looking to improve the quality of the updates to the IT systems. For example, the Waterfall approach is to be substituted by an Agile one to improve the overall status quo of software development.

4.1.2 Sub phase 2 - Set Context

The objectives proposed for this dissertation fit within the business strategy of having a capable, flexible and maintainable tool to process every IT related aspect of the operation. These attributes are a pre-condition for the core system transition to happen successfully with the sunset of the legacy one.

4.1.3 Sub phase 3 - Build Sponsorship

Considering the problems found in chapter 3. Problem Analysis, there are valid reasons to endorse change as an improvement tool. The survey and presentations produced served as a channel to create awareness and engagement to these problems. In particular, the survey had a stronger adherence from the management layer when compared to the staff layer, being the first critical early in the process (SEI 2009). However, this stronger adherence only represents one-third of the management layer.

A final presentation, containing the problems found and proposed solutions is to be performed in the future to both Deloitte and the client personnel. This future moment is critical in propelling the previously generated awareness into action.

4.1.4 Sub phase 4 - Charter Infrastructure

To have the required resources to set up the mechanisms for managing the implementation details during the effort, it is necessary to have committed part of the organizations' personnel. If the change consists in a broad and complex effort, it may require 2-3% of the organization's people (SEI 2009).

Since the actual acting activities are out of scope for this dissertation, no further considerations will be done concerning this topic.

4.2 Diagnosing Phase

In this section, an As Is and a To Be characterization of the organization is presented to build an approach for improving business practice.

4.2.1 Sub phase 5 - Characterize Current and Desired State

The whole chapter 3 is dedicated to the first part of this point, As Is. In this exploratory analysis gaps within defect reporting and communication were discovered, and synthesized in Table 10. The CMMI template presented at Appendix E will be used to define the desired state and compare it to the current state.

The current state fits within the CL - 2 MANAGED, despite having some level CL -3 DEFINED traits. There are established organizational policies to regulate processes. As such, activities are planned and executed accordingly by skilled employees to produce controlled outputs, while being monitored, controlled and reviewed to evaluate the adherence to process description. In Table 11 is presented a critical analysis of the current state of each Generic Practices.

Table 11 – Current and desired state comparison – CL - 2 MANAGED and CL - 3 DEFINED

Generic Practices	In Force	To Be	Comment
GP2.1 – Establish Organizational Policies	Yes	Yes	Meetings with Deloitte’s professionals concerning the state and prospects of the whole transformation project
GP2.2 - Plan the Process	Yes	Yes	Annual plan with all the initiatives to take place
GP2.3 – Provide Resources	Yes	Yes	Access to the client’s systems and tools, to How to documents and appropriate facilities and online meeting tools
GP2.4 – Assign Responsibilities	No	Yes	Currently the dynamic assignment of tasks works, but it does not cover specific but relevant aspects of the process
GP2.5 – Train People	No	Yes	In particular, with testers, proper train can be neglected resulting in the misclassification of expected behavior as a defect. Through workshops, for example, it is possible to deliver the necessary skills and knowledge so testers can perform their roles effectively and efficiently
GP2.6 – Manage Configurations	No	Yes	The classification itself is not allowing for an effective control of the work products, namely defects, as relevant information is sometimes absent
GP2.7 – Identify & Involve Relevant Stakeholders	No	Yes	Information presented and decisions take at high-level meetings are not shared with the elements performing the work
GP2.8 – Monitor and Control the Process	Yes	Yes	Senior team elements cover the scope of this GP
GP2.9 – Objectively Verify Adherence	No	Yes	Incomplete adherence to the documented guidelines
GP2.10 – Review Status with Higher Level Management	Yes	Yes	Higher level management is involved in the process, receiving relevant information to make informed decision on the plan
GP3.1 – Establish a Defined Process	No	Yes	By complementing the documented procedures with the ones presented in the next section, 4.2.2, the relevant processes related to defect handling from a tester point of view are covered
GP3.2 – Collect Process Related Experiences	No	Yes	With the standardization of classification, the collection of relevant information for improvement is made simpler and easier

The purpose of this dissertation is to improve the software classification process closer to the CL - 3 DEFINED. Thus, the To Be model objective is to guarantee that there are established defined processes and that information on improvements is being collected.

4.2.2 Sub phase 6 - Develop Recommendations

The present chapter is dedicated to the presentation of improvement suggestions. These recommendations have been thought so as to tackle the gaps identified under the chapter 3 - Problem Analysis.

Roles and Responsibilities

In Table 12 is presented the result of mapping the main testing activities and team roles. To each activity and role is defined a degree of responsibility.

Table 12 – Roles and responsibilities in testing

Activity	Role				
	Functional			Technical	
	Test Manager	Test Designer	Tester	Develop. owner	Developer
P – Primary (performs the activity tasks) S – Secondary (engages in some part of the activity) A – Aware (conscious of the activity progress) - – None					
Planning					
- Determine scope, resources, risks, objectives, approach (manual or automated)	P	S	-	S	-
Control					
- Monitor and document progress	P	-	A	A	-
- Share information on testing status	P	-	S	S	A
Designing					
- Identify test conditions	-	P	A	-	-
- Evaluate testability and requirements	A	P	A	S	-
- Create test cases	S	P	-	A	A
- Review test cases	-	S	A	S	P
Preparation					
- Create Test Suites	P	-	A	A	A
- Workshop to testers and developer on relevant business knowledge for the requirements under test	P	-	S	P	S
- Share improvements to the testing processes	P	-	S	P	S
- Assign specific test	P	-	S	A	A
Execution					
- Execute test cases and retest	A	-	P	A	A
- Correct test scripts	A	S	P	A	-
- Fix defects	A	-	A	S	P
- Details on deployments	A	-	A	P	S
- Share workarounds and testing shortcuts	S	-	P	S	P
- Ensure effective communication	P	-	S	P	S
Reporting					
- Open defects	A	-	P	S	A
- Follow-up on defects	S	-	P	S	P

From the activities listed in Table 12, the ones concerning sharing information are not currently attended effectively. These gaps generate significant communication problems as relevant information is not being shared with every interested party.

To engage both functional and technical teams into working together more effectively, a moment to review the collective performance on each test phase would be useful. Also, by promoting scrum⁹ meetings with both teams to establish teamwork solid relationships and share knowledge, difficulties and the current progress of tasks, it is likely for the communication to improve.

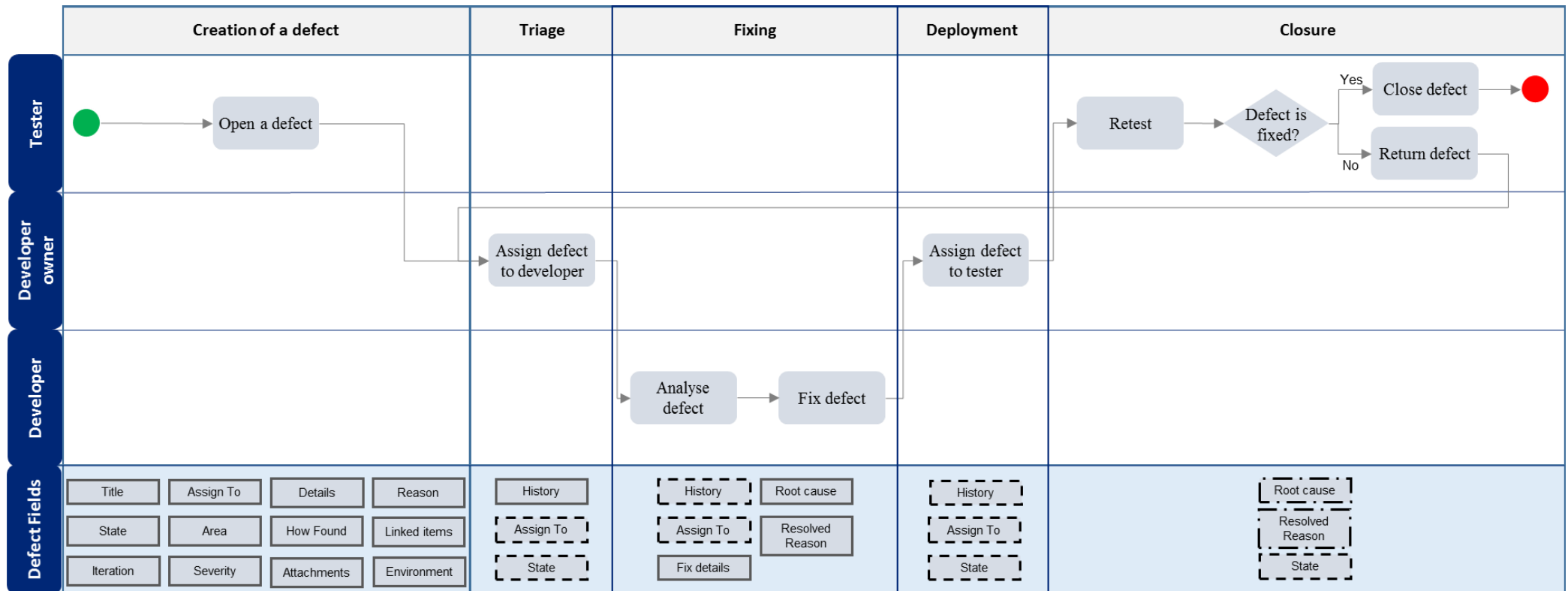
Also, it is relevant to note that many times the testing teams are assembled to share these roles and responsibilities between its participants. Thus it must serve more like a flexible guide than a hard restraint. It is important to be aware of these activities and to define within each team which roles each element is going to play.

Figure 14 presents the responsibilities in filling the defect fields during its life cycle. With this mapping, every element of the defect handling process can easily understand what fields he is supposed to fill in, change or validate.

Also, in rare cases, a defect may be opened without associating it to a test case. For instance, when abnormal behavior is detected while executing a test, but it is not covered in its scope. In this case, the repro steps are not filled in by TFS, as it only happens if the defect was created while executing a test case. Thus, if the defect is being opened without an associated test case, the field details must contain which steps to follow to reproduce the defect.

As a final remark, team workers must commit every effort into simplifying communication, as a breakdown in the chain linking team members leads to data loss, need for rework, lack of clarity and ultimately lower software quality. This principle applies not only when interacting with different teams, but also within teams. By knowing the responsibilities related to this process and filling the right fields with proper information, the whole collaboration process is made easier.

⁹ Scrum – mandatory team brief meetings where each team member presents the progress of his work and any eventual impediments in his way



Legend:



Figure 14 – Defect fields responsibilities: life cycle overview

Testing

In section 2.3.2, necessary information is provided on how to find a good trade-off between lower quality and delaying the release of the software. From observation, the relevance of the contents with the reality at this project is noticeable, since testing, from observation, is easily susceptible to be under heavy pressure. On the other way around, the repetition of test cases and disuse of shared steps functionality, analyzed at section 3.4.1, generate wasted work. Thus, opportunities to generate more tests aimed at problems found in previous testing phases are lost.

Furthermore, concerning UT they are performed solely by the developer. If an element of the functional team took part of the UT to assess if a minimum coverage of the requirement was being attended, defects could be prevented up in the development phase to avoid increased downstream costs, as presented in Figure 2.2.4.

Also, as stated in section 3.4.1, there is no established practice to use the information on SharePoint PRIORITY field for the requirement under test to establish how deeply it must be tested. If assumed that to a higher PRIORITY, which means higher urgency to be implemented, is associated an increased concern with the requirement quality, then it would make sense to create test cases considering this information.

Concerning tests execution, testing requires skill and knowledge on the application being tested, particularly when writing the test cases. The latter should be written with as much detail as possible, to enable their execution by less knowledgeable elements of the team. However, due to time constraints, it may not be possible to put the necessary effort to build solid test case steps and thus detail is sacrificed. This type of risks should be taken into consideration when defining the team's constitution, so as to counter the lack of detail with experienced testers. Adopting the tactic of having a technical element review the test cases can be of great value, not only to address incorrect testing procedures, but also to improve the detail by bringing technical insights.

In the particular case of UAT, it is reasonable to consider having Deloitte's personnel filtering the defects opened by client's testers since the analysis revealed a relatively higher percentage of defects not ending up as FIXED. This Deloitte's member would review reported defects before they go to the developer, checking that critical information is present and that it is possible to reproduce the defect. If there are problems, he must take the defect back to the original reporter justifying the decision.

To finalize this section, the tester mindset should be one not to limit quality control to how the requirement was designed, but instead to the relevance and impact on the business it was meant to have. In particular, when reporting defects, relevant data to allow for an informed business decision on the resolution of the defect must be a priority, especially when it is a suggestion or a likely incident.

Functional Design

Testers execute and design tests considering information present mainly at functional design documents, in other words, functional design defines expected behavior. In the survey, misclassification was associated with incomplete/flawed FD. It was even stated by elements of the technical team that ‘Before opening a defect, we need to ensure we know the expected behavior (not only based on Functional Documentation).’

Considering the observation that testers are not necessarily present in the meetings with the client and often do not take part in the design phase, as stated in section 3.4.1 - Testing. It should not be expected more than a critical analysis of the expected behavior defined in the test case and related documentation to decide if a certain behavior is a defect or not. Workshops organized by the technical team to explain how the requirements have been implemented can be useful to tackle both situations when there are technical limitations in the implementation and when the functional design is flawed.

At section 4.2.2 – Roles and responsibilities, with the definition of roles and responsibilities among team workers, it is expectable that contingencies to address functional design gaps can provide a better understanding of the real expected behavior.

Work Items

At the moment to report software defects only the work item defect is being used. Moreover, this is also used to address test environment problems that are not the result of problems with developments but instead are related to integration services unavailability.

To address these issues, in Table 13 is presented the proposed list of work items containing their title, description, and relevance.

Table 13 – Work items list: defect and impediment

Title	Description	Relevance
Defect	When a defect in the system’s behavior is detected based on FD and TD	Cover problems with non-conformity with design
Impediment	Errors related to unavailability of the integration layer. See note (1).	Possible to track the impact of integration layer unavailability on the testing progress

Note:

- (1) – If, after checking with the technical team the problem to be reported, the issue is related to integration unavailability an impediment should be created. Note that this concept is especially applicable for SIT phase.

Since TFS 2013 includes the work item impediment, the only matter to address is the engagement and agreement required from functional and testing teams to put this concept into practice.

Root Causes

Concerning ROOT CAUSE, it is relevant to note that its importance is tied to understanding the weaknesses in the design, development, and testing. As observed in section 3.4.2, most of the time this field is left blank and when it is not, CODING ERROR is by far the most common cause. Given that this dissertation focuses on the functional aspects of the process, the technical reality was not covered. Still, there are some key points, listed below, worth of attention for future reference:

- lack of technical expertise from the developer;
- incomplete code review – since the code will be tested and due to time constraints, this condition may be relaxed.

Also, the ROOT CAUSE list cover only situations where the defect reported was an actual defect. Since there is a substantial percentage of misreports, the addition of ROOT CAUSE REPORTING ERROR improves the consistency of the defect classification process.

Furthermore, the quality responsibility should not be enforced in testing, but in the development process. The current testing methodology follows a Waterfall logic that does not entirely engage the developer in the quality of the software produced, thus not creating a commitment to assure quality at the source. Some responsibilities defined at team level can help mitigating this issue, as suggested in section 4.2.2 – Roles and responsibilities.

State

Concerning the field STATE, the addition of a new state DEFERRED would allow covering situations where the defect resolution was postponed to a future release. This way, not only it is now possible to quickly search for defects in this situation, as there is no reason to open defects that are known and to be fixed in a future moment.

Severity

It was observed that the actual practices concerning this field do not follow the established guidelines documented at Deloitte's eRoom, as the default is 3-MEDIUM with exceptions. However, it does not have a significant impact on the defect handling process. On the one hand, every defect must be fixed, and on the other hand, technical team does a smart allocation of its available resources in order to minimize the impact on the progression of tests. As a result, it is not a relevant problem if a problem at all.

Considering that defects may be associated not with the developments under test, but to defects present in production, it is pertinent to attribute this cases a new level 6 - INCIDENT. This way, when a given test fails because of something that is not reportedly in the scope of the implementation of a requirement, the defect should be created with Severity 6 - INCIDENT, working as a potential incident report.

A clarification between both these concepts is relevant. Concerning 5 - SUGGESTION, by definition, it must be a case of something that has been implemented according to the design but is not aligned with business needs. If it is a 6 - INCIDENT, then the implementation is not according to the design but has not been detected and fixed in the correspondent release.

Reporting Defects

The actions presented in Figure 15 illustrate how the tester must proceed when faced with potential defects, to complement Figure 11.

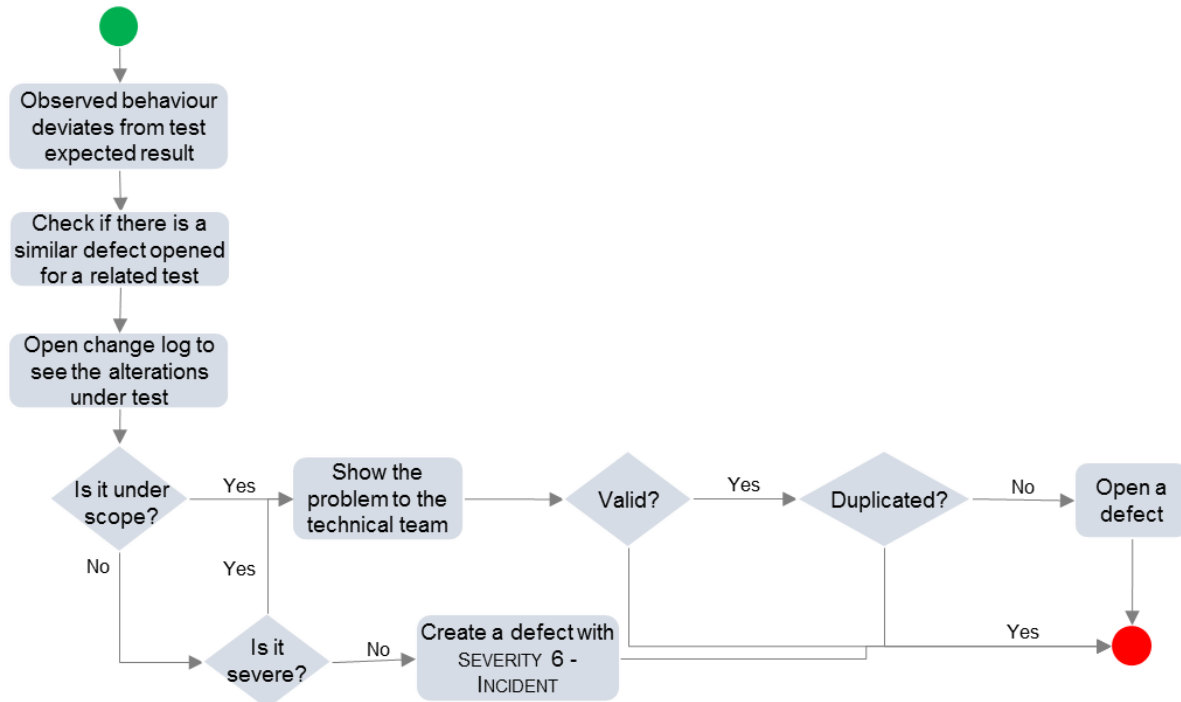


Figure 15 – Flowchart of activities to follow before opening a defect

Note that the new SEVERITY 6 - INCIDENT is being applied to handle problems outside of scope for the new developments. However, if the problem found is severe, meaning that it would rank level 1 - CRITICAL or 2 - HIGH in SEVERITY scale, it should be presented to the technical team. This way, the escalation of this defect to the client's responsible may be accelerated to assess if it really is a problem in production and, if so, an incident open to address the issue.

A defect report must be limited to a single defect as multiple defects inside the same defect corrupt its content and generate unwanted complexity. Moreover, it compromises the ability to easily collect relevant process improvement data and with simple procedures. A defect, if transversal to multiple test cases, must be associated to each one of them as long as the reported behavior is aligned.

Resolved Reasons

The first thing to do concerning this topic is to define each RESOLVED REASON properly to uniformize the concept across teams. In Table 14 is presented relevant information on the RESOLVED REASON subject.

Table 14 – RESOLVED REASON: definition and comments

RESOLVED REASON	Definition	Notes
FIXED	The described issue has been resolved	-
AS DESIGNED	Behavior reported is according to design	Take into account the possibility of mistakes in documentation, this is the main reason why design must always be critically interpreted
DUPLICATE	Multiple reports of the same defect under the same testing phase	This resolution should be set from a technical point of view, as the same issue may cause multiple behavior deviations
REJECTED	Behavior reported is the expected one, but there is no design to support the claim, or when defects are result of problems in the environment	Rejected is used to cover the particular situations when a defect is wrongly reported because the design is incomplete but the implementation is correct, or the environment is not working as supposed
CANNOT REPRODUCE	The developer could not reproduce the problem. Thus the investigation stopped	Some issues may only appear under very specific conditions, which are oblivious to the tester. This is the baseline for this classification, since if there is no sufficient data or the developer does not understand the repro steps, clarifications with the reporter must be requested

The objective is not to have always FIXED in the RESOLVED REASON field, but to be able to find a good compromise between accuracy and time invested in investigating potential defects.

Even though not all the misreported defects analyzed in 3.4.2 – Resolved Reason: Causal Analysis - had a justification for their resolution, when looking at FIXED ones it is highly common for no comment to be provided in this classification. Since FIXED means that through investigation the issue reported was a real problem and a fix has been made available, it is relevant to provide information on the fix employed. This way, in the future, if a similar defect occurs the developer could start by performing simple queries to look for past similar defects if the problem is not known to him and does not appear to be related to recent developments.

Given that this behavior results in extra work for the developer, it should only be performed when it is a defect with a significant likelihood of re-occurrence as, for example, problems with special characters that can appear in multiple areas of the program. From observation and historic analysis, it is possible to see some cases of repeated defects in different moments. Since DUPLICATE is applied when the same defect is reported multiple times in the same testing phase, these are deemed as new problems, even though they occurred in the past.

To sum up, the developer after fixing a defect should assess the likelihood of its reoccurrence, due to its nature, and thus provide details on its fix so as to provide useful information on its resolution to be taken advantage off.

Resolved Reasons: Root Causes

As observed in section 3.4.2 – Aggregated analysis: Overview, a considerable effort is spent handling defects that are not defects. As such, it is relevant to understand how improvements can be introduced into the teams' practices to make defect management more efficient and effective.

The high percentage of defects that end up with a resolved reason different than FIXED is not the problem in itself, but a symptom of a problem in the reporting process and communication flows.

In Table 15 are listed the root causes identified in sections 3.4.2 and 3.4.3 for the attribution of a RESOLVED REASON other than Fixed, through the analysis of historic and the survey.

Table 15 – Proposed improvements to address faults in defect reporting

Root Cause	Proposed improvements
'As Designed'	Clarification on RESOLVED REASONS - see Table 14, section 4.2.2 – Resolved Reasons
Technical Limitations	Workshops organized by the technical team - See section 4.2.2 – Functional Design
Incorrect testing procedure	Test scripts review - See section 4.2.2 - Testing
Replicated defect report	Reporting a defect procedure - See section 4.2.2 – Reporting Defects
New requirement / valid suggestion	Reporting a defect procedure and SEVERITY considerations - See section 4.2.2 – Severity and Reporting Defects
Waiting for deployment	Information on environment updates - See section 4.2.2 – Roles and Responsibilities
Deferred defect	New defect state DEFERRED - See section 4.2.2 - State
Different testing conditions	See note (1)
Flawed functional design	Functional design considerations - See section 4.2.2 – Functional Design
Available workaround	KT on workarounds - See section 4.2.2 – Roles and Responsibilities
Unclear/absent repro steps	Awareness of good reporting practices - See section 4.2.2 – Roles and Responsibilities
Lack of business knowledge	Train testers in the major business aspects - See section 4.2.2 – Roles and Responsibilities
Doubts in the defect classification	Reporting a defect procedure and functional design considerations - See section 4.2.2 – Functional Design and Reporting Defects

Note:

- (1) – This cause was associated with two main reasons, unavailability at integration layer and updated environments. For the first one, there is new work item type impediment, presented in section 4.2.2 – Work items. As for the second one, its cause is very similar to Waiting for deployment, in more detail at section 4.2.2 – Roles and responsibilities

In the sample of defects studied, it was observed that the information provided is ad-hoc and does not allow a quick understanding of the root cause for the RESOLVED REASON choice.

Thus, by providing a list to cover the most common reasons it is possible to gather information easily in future releases on the causes for the creation of misreported defects. The suggested list corresponds to the first column of Table 15.

Continuous Improvement

Section 4.2.2 addresses the gaps in the defect handling processes found at chapter 3 Problem Analysis, thus establishing defined processes for every relevant activity or, in other words, standardizing classification.

Through the standardization of classification, valuable information with many useful applications is made readily available. Since that the earlier a problem is found within the software life cycle the cheaper it can be fixed, the usage of tools and methodologies, such as causal analysis, to prevent and find problems sooner becomes valuable. For instance, this improvement information can identify which gaps are causing the introduction of most errors, in this particular case, classification ones, and assist in evaluating the quality of the software developed.

4.3 Establishing Phase

In the previous section, 4.2.2, the recommendations contain the details to achieve CMMI Capability Level 3. For this section, a hierarchy of improvement suggestions and considerations related to the approach to their implementation is presented.

4.3.1 Sub phase 7 - Set Priorities

In Table 16 is presented a compiled list of the improvement suggestions made at section 4.2.2, associated with a priority. The suggested hierarchy is based on the dependencies between recommendations.

Table 16 – Hierarchy of improvement suggestions

Priority	Improvement Suggestions
1	- Standardization of classification – solidly defined process to cover every aspect of defect classification and handling
2	- Roles and responsibilities – specific functions and responsibilities at every step of the testing process - Testing procedures – reduce time spent in unnecessary work and improve risk-based approach trade off
3	- Continuous improvement – collect information on previous performance and main issues to serve as input for improvement

Starting with the definition of standard classification norms, follows the endorsement of roles and responsibilities to guarantee their adherence and effective team communication. In other words, after defining the process, responsibilities regarding its core aspects are assigned to the team workers.

Afterward, the conditions to collect information for continuous process improvement are set. Thus, the related procedures of data extraction and analysis must be put into practice.

4.3.2 Sub phase 8 - Develop Approach

Even though the CMMI software development guidelines present an interesting route to plan and structure organizational improvement, it requires a considerable amount of effort from the organization to be put in a path to high levels of capability and deep commitment at every hierarchic level. Also, it demands that certain assumptions related to non-chaotic behavior be made concerning the organization to model the specific practices (Beedle et al. 1999).

In this particular case, given Deloitte teams' frequent changes and the turnover of professionals between different projects does not condone with stability. Due to schedule pressure, priorities and contingencies the modeled processes must be flexible. On the other hand, the documentation produced in the process would likely be too dense to use it to train people effectively.

Still, even if the client took the initiative to commit to achieving high levels of software development capability, part of the scope required to endorse these changes is out of their reach since they are using an external core system with limitations due to its very architecture.

As a result, it should be used to characterize the desired state or, in other words, the To Be model, but not as a roadmap to improvement. For this purpose, courses, workshops and knowledge transfer through the eRoom and SharePoint are the main vehicles to deliver the recommendations made during the diagnosing phase.

4.3.3 Sub phase 9 - Plan Actions

With the approach defined and the priorities established, a detailed implementation plan is in condition to be created. This plan must include schedule, milestones, responsibilities, measurement metrics, risks and mitigation strategies, among other elements relevant for the implementation.

Since that the developed recommendations are not to be implemented in the scope of this dissertation, no further analysis of this topic will be made.

4.4 Acting and Learning Phases

The previous approach holds little value if not put into practice. To create awareness, multiple presentations were performed and a survey was shared to gather opinions from different people and engage them in this improvement effort.

The activities of the acting phase help an organization implement the work that has been conceptualized and planned in the previous three stages and once the solution is workable, it can be applied throughout the organization. With the learning phase ends the improvement cycle, which objective is to review what was accomplished and how can the organization in future change efforts be more successful in its improvement initiatives, starting the cycle again.

5 Conclusions and Future Work

In a real world problem, frequently there is no readily available data to be worked on and hard assumptions result in significant deviations from reality. Still, simplifications of the problem can be made as long as they are known and understood. Moreover, uncertainties are part of the daily organizational life. Thus adaptive approaches can handle chaos more effectively and provide a better assessment of risk and alternatives.

As organizations are extremely complex, due to their culture, people and established working procedures, every improvement effort to be put into practice implies that conditions for its success are well identified and linked together. Deloitte's professionals were asked to participate in presentations and to answer a survey in order to create awareness and engagement to the matters here studied.

Even though the initial purpose of the project was to create a new classification system, the problem on misclassification is mostly a symptom, not a root cause. Thus, the exploratory work, which included observation by being present in the field, historic analysis to understand previous practices, a survey to capture insights from the large group of people involved in this project, revealed deviations from defined good practices and communication problems.

For example, it was observed in Resolved Reason field that one-third of the defects opened in the past was not actual defects, which represents a considerable inconvenience for technical teams as time is spent investigating reports that are not defects.

Moreover, communicating with the technical team before opening a defect may be a red flag that suggests flaws in the way requirements are documented, even though, in this context, it can be considered a good practice. Most of the answers obtained in the survey were from the management layer of the firm, which means that these professionals seem more interested in improving the defect handling when compared to those who perform the work in the field.

As for the proposed objectives, the developed standard framework for classifying software defects addresses gaps identified in the activities associated to this process. It is also suggested how the mapping of defect information can be made so as to allow relevant improvement information, such as root causes for misreporting of defects, to be easily collected.

With the definition of roles and responsibilities, work can be shared more effectively and communication issues mitigated. Also, the created sense of integrated environment and clarification on which fields are to be filled in each main step of the defect handling process promotes high integrity in the reporting of defects.

Relative to the objective leaner testing procedures, it was observed that repeated and redundant tests were executed. Thus, time is spent on rework rather than performing additional relevant tests. Moreover, starting proper testing as soon as possible allows for a reduced cost on early defect fixing, as expressed in Figure 4. Promoting the adoption of criteria to prioritize the quality importance of different requirements enables a more effective attribution of testing resources.

Improved system quality is achieved as a result of the success in the previous objectives. Namely, through the standardization of classification practices, it is expectable that the number of defects reported that are not actual defects is reduced allowing the developer to focus on real problems. With roles and responsibilities, the teams are better protected against broken communication links and consequent loss of valuable information. Having a leaner testing procedure allows for more tests to be performed, thus potentially increasing the number of reported defects and ultimately leading to enhanced quality.

The system quality is not limited to conformity with requirements or, in other words, functionality, but also related to other factors. Concerning usability and efficiency, these aspects are improved as a result of better functionality and less unexpected behaviors. About maintainability and adaptability, as a consequence of the improved defect classification and testing procedures, the sunset of the legacy core system becomes less risky. Thus, the new system with higher flexibility and lower cost can take its place with increased ease.

To sum up, the improved model covers communication, classification practices, testing methodology as well as roles and responsibilities areas of improvement to upgrade the CMMI capability from level 2 - Managed to 3 - Defined. With the upgrade, work can be conducted more consistently and in a continuous improvement basis, essentially identifying faults and capturing best practices.

A future presentation is scheduled, meant to both Deloitte and the customer, to expose the new framework to follow for defect classification. This presentation will be a first step for the Acting and Learning phases presented in Figure 13, as the relevance of the presented suggestions is tied to their adherence. Thus, in the future, effective change management must be put in place so that the required commitment and engagement for the improvement effort may be successfully achieved.

References

- Basili, Victor R, and Gianluigi Caldiera. 1995. "Improve software quality by reusing knowledge and experience." *MIT Sloan Management Review* no. 37 (1):55.
- Beedle, Mike, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland. 1999. "SCRUM: An extension pattern language for hyperproductive software development." *Pattern Languages of Program Design* no. 4:637-651.
- Bettenburg, Nicolas, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. "What makes a good bug report?". Paper presented at Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering.
- Boehm, Barry, and Victor R Basili. 2005. "Software defect reduction top 10 list." *Foundations of empirical software engineering: the legacy of Victor R. Basili* no. 426.
- Chen, Pei-Chi, Ching-Chin Chern, and Chung-Yang Chen. 2011. "Towards An Integrated Effort For Managing IT Process Standards Implementation". Paper presented at PACIS.
- CMMI. 2016. "Begin Your CMMI Journey". Accessed 2016/05/24. <http://cmmiinstitute.com/get-started>.
- Dawson, Maurice, Darrell N Burrell, Emad Rahim, and Stephen Brewster. 2010. "Integrating Software Assurance into the Software Development Life Cycle (SDLC)." *Journal of Information Systems Technology and Planning* no. 3 (6):49-53.
- Deloitte. 2006. *How Insurance Companies Can Beat the Talent Crisis*.
- . 2014. "Brochura Institucional."
- . 2015. "Tech Trends 2015 - The fusion of business and IT: An Insurance Industry Perspective".
- . 2016. "Learn about our global network of member firms". Accessed 2016/02/25. <http://www2.deloitte.com/pt/en/pages/about-deloitte/articles/about-deloitte.html>.
- Graham, Dorothy, Erik Van Veenendaal, and Isabel Evans. 2008. *Foundations of software testing: ISTQB certification*: Cengage Learning EMEA.
- Hoyer, Robert W, Brooke BY Hoyer, Philip B Crosby, and W Edwards Deming. 2001. "What is quality?" *Quality Progress* no. 34 (7):52.
- IEEE. 2010. *IEEE Standard Classification for Software Anomalies*.
- ISO. 2011. ISO/IEC 25010:2011. *Em Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models*.

- Juran, Joseph M, and A Blanton Godfrey. 1999. "Quality handbook." *Republished McGraw-Hill*.
- Kaner, Cem. 2002. "Bug Advocacy". Paper presented at Quality Week 2002.
- Kumaresh, Sakthi, and R Baskaran. 2010. "Defect analysis and prevention for software process quality improvement." *International Journal of Computer Applications* no. 8 (7).
- Kumaresh, Sakthi, and Baskaran Ramachandran. 2012. "Defect Prevention Based on 5 Dimensions of Defect Origin." *International Journal of Software Engineering & Applications (IJSEA)* no. 3 (4).
- Marr, Bernard. 2015. "How Big Data Is Changing Insurance Forever." *Forbes*. <http://www.forbes.com/sites/bernardmarr/2015/12/16/how-big-data-is-changing-the-insurance-industry-forever/#255d6426435e>.
- Microsoft. 2010a. "Using Team Foundation to Manage Development Processes". Accessed 2016/05/28. <https://msdn.microsoft.com/en-us/library/ee854733.aspx>.
- . 2010b. "Using the Team Project Portal". Accessed 2016/05/28. [msdn.microsoft.com/en-us/library/ms242883\(v=VS.90\).aspx](https://msdn.microsoft.com/en-us/library/ms242883(v=VS.90).aspx).
- . 2010c. "Visual Studio Application Lifecycle Management". Accessed 2016/05/28. <https://msdn.microsoft.com/en-us/library/fda2bad5%28v=vs.100%29.aspx>.
- . 2012. "Organizing Test Cases Using Test Suites". Accessed 2016/05/29. <https://msdn.microsoft.com/en-us/library/dd286738%28v=vs.110%29.aspx>.
- . 2015a. "Branch strategically". Accessed 2016/06/18. <https://www.visualstudio.com/en-us/docs/tfvc/branch-strategically>.
- . 2015b. "CMMI process work item types and workflow". Accessed 2016/05/28. <https://www.visualstudio.com/docs/work/guidance/cmmi-process-workflow>.
- . 2015c. "Running manual test using the web portal". Accessed 2016/05/29. <https://msdn.microsoft.com/en-us/library/dd286725.aspx>.
- . 2015d. "Submit a Bug Using Microsoft Visual Studio". Accessed 2016/05/29. <https://msdn.microsoft.com/en-us/library/dd293538.aspx>.
- Ryu, Marcus. 2007. "Reasons To Replace Legacy Systems." *Best's Review* no. 108 (1):82-84. <http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=25094186&lang=pt-br&site=ehost-live>.
- SCHAEFER, Hans. 2006. "Strategies for Prioritizing Tests against Deadlines Risk Based Testing." *Software Test Consulting, Noruega*, <http://home.c2i.net/schaefer/testing/risktest.doc>, acesso em no. 20 (11).
- Schaefer, Spillner Linz. 2014. *Software Testing Foundations 4th Edition*.
- SEI. 2007. *Capability Maturity Model Integration (CMMI) Version 1.2 Overview*. Carnegie Mellon University.
- . 2009. *The IDEAL Model*: Carnegie Mellon University.
- . 2010. *CMMI for Development, Version 1.3*. Carnegie Mellon University.
- Zaineab, Ghazia, and Irfan Manarvi. 2011. "Identification and analysis of causes for software bug rejection with their impact over testing efficiency." *International Journal of Software Engineering & Applications (IJSEA)* no. 2 (4):12.

APPENDIX A:

Testing defects root causes for defective coding (Kumaresh and Ramachandran 2012)

Pain Points	Root Causes	Defect Prevention
Project Leader's Inability in Resource Allocation	Inefficient Work Break Down (WBS) structure	PM should review the WBS assigned by the PL and should take mitigation steps if he foresees any risk in it.
Lack of Technical Expertise	Project team members are not trained properly (or) may not have worked in similar technical domain.	Skilled resources with Technical work experience should be inducted into the project, if project has tight deadlines.
Missing Coding Artefact	Due to shortage of time	PL should find reasons for time constraints and should talk with PM to see if the estimated effort has to be relooked for sufficiency
Non adherence to coding standards	To complete the code fast and due to time constraints, developers may not adhere to coding standards	PM/Quality Team can cross check if PL is doing the code review or not.
Incomplete code review	PL might feel that testing the code would suffice and may skip the review.	Quality Team can do audits / checks to see if review is done formally.
Coding Errors	Programmers not familiar in developing similar type of codes or errors may be due to careless mistakes	Training should have been provided before the development starts.

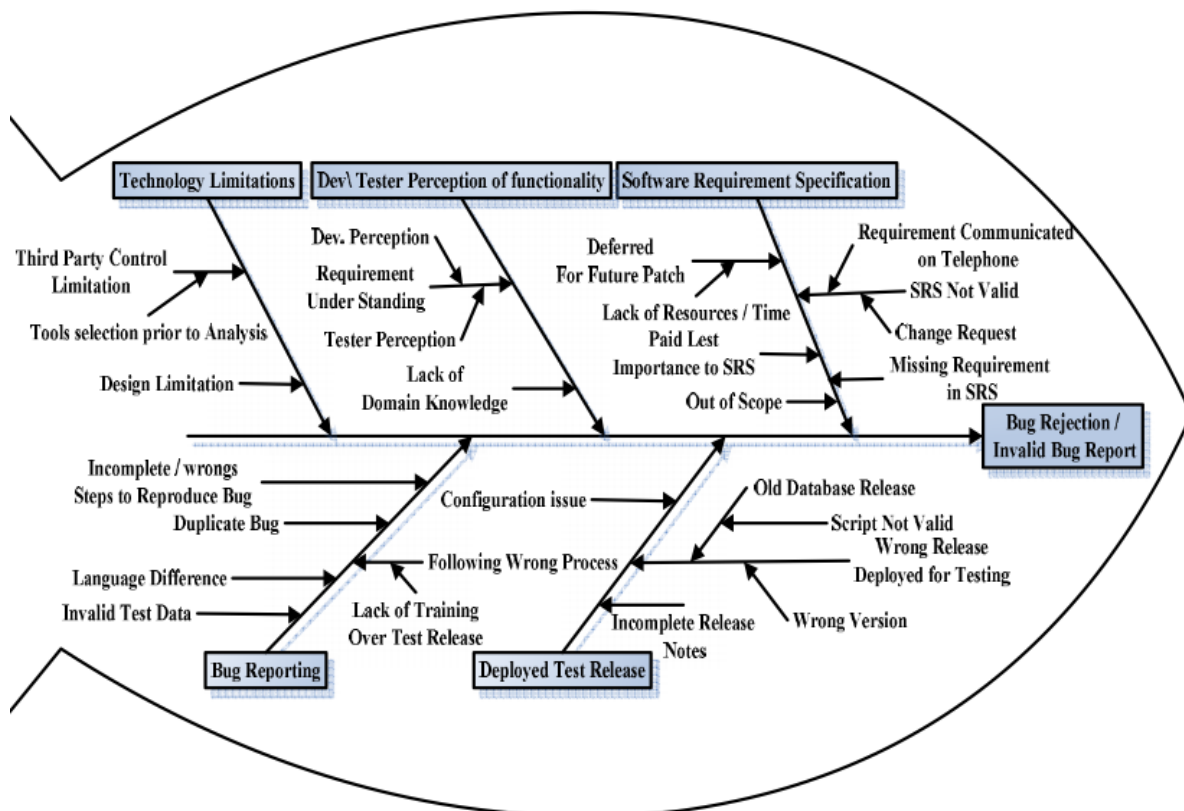
APPENDIX B:

Testing defects root causes for delinquency in testing (Kumaresh and Ramachandran 2012)

Pain Points	Root Causes	Defect Prevention
Inadequate test cases	The testers may lack knowledge on the possible scenarios of test path for the requirements.	The BA should run-through all possible scenarios of the actual Business to the Test team so that the Testers can build a strong Test Case base for the project
Pretermision	Due to lack of time and schedule slippage, pressure might mount on the team to move to the next level of testing.	The PM shouldn't hide the facts as up to what level the Testing is performed and should involve the client for a joint decision.
Dissimilar test Environment	Space restrictions in the Test machine	Space restrictions, if any, should be handled by PM and client mutually.
Uncovered test bed simulation	Unavailability of Database space may restrict the Test bed setup.	Size of database required for testing has to be planned (by PM & PL) upfront and sufficient time should be given to the IT Team for DB space allocation.
Eclipses in testing	Due to unavailability of the Testing tool S/W, Experienced Testers not available.	While planning the project, the PM should decide (along with the client) how these tests can be performed Testers with expertise in such type of testing can be used from other teams in the Organization
Shortening the testing time	Inadequate testing	Proper Planning should be done before start of SDLC activities.

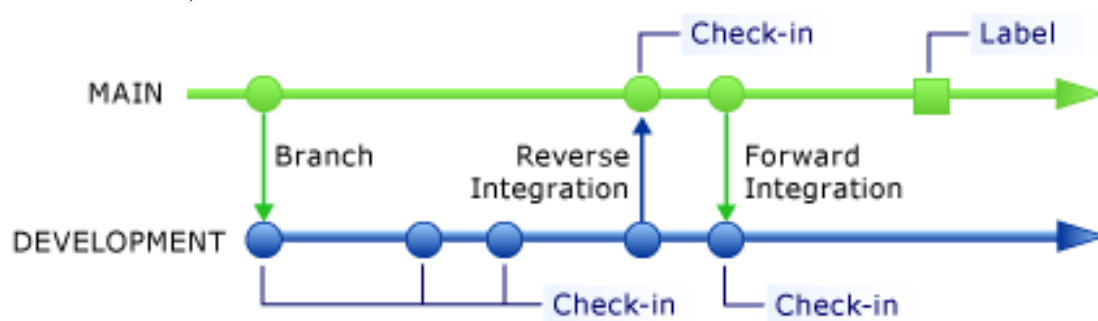
APPENDIX C:

Cause and effect Figure for defect rejection (Zaineb and Manarvi 2011)



APPENDIX D:

Branching strategy can be used to isolate development branches from the main application, thus allowing for concurrent developments while keeping the main application stable and protected (Microsoft 2015a)



APPENDIX E:

Organizational Capability Levels – adapted CMMI design (Chen, Chern, and Chen 2011)

Capability Level	Generic Goal	Generic Practices	
0	N/A	No requirements for software development institutionalization	
CL-1 (Incomplete)	GG-1: Achieve specific goals	GP1.1	Perform specific practices
CL-2 (Managed)	GG-2: Institutionalize a managed process	GP2.1	Establish organizational policies
		GP2.2	Plan the process
		GP2.3	Provide resources
		GP2.4	Assign responsibilities
		GP2.5	Train people
		GP2.6	Manage configurations
		GP2.7	Identify & involve relevant stakeholders
		GP2.8	Monitor and control the process
		GP2.9	Objectively verify adherence
		GP2.10	Review status with higher level management
CL-3 (Defined)	GG-3: Institutionalize a defined process	GP3.1	Establish a defined process
		GP3.2	Collect improvement information
CL-4 (Quantitatively Managed)	GG-4: Institutionalize a quantitatively managed process	GP4.1	Establish quality objectives for the process
		GP4.2	Stabilize sub-process performance
CL-5 (Optimizing)	GG-5: Institutionalized an optimizing process	GP5.1	Ensure continuous process improvement
		GP5.2	Correct root causes of problems

APPENDIX F:

Detail on each specific generic practice for CMMI Capability Level CL-2 Managed (SEI 2010)

GP 2.1 **Establish an Organizational Policy**

Establish and maintain an organizational policy for planning and performing the process.

The purpose of this generic practice is to define the organizational expectations for the process and make these expectations visible to those members of the organization who are affected. In general, senior management is responsible for establishing and communicating guiding principles, direction, and expectations for the organization.

Not all direction from senior management will bear the label “policy.” The existence of appropriate organizational direction is the expectation of this generic practice, regardless of what it is called or how it is imparted.

GP 2.2 **Plan the Process**

Establish and maintain the plan for performing the process.

The purpose of this generic practice is to determine what is needed to perform the process and to achieve the established objectives, to prepare a plan for performing the process, to prepare a process description, and to get agreement on the plan from relevant stakeholders.

The practical implications of applying a generic practice vary for each process area.

For example, the planning described by this generic practice as applied to the Project Monitoring and Control process area can be carried out in full by the processes associated with the Project Planning process area. However, this generic practice, when applied to the Project Planning process area, sets an expectation that the project planning process itself be planned.

GP 2.3 Provide Resources

Provide adequate resources for performing the process, developing the work products, and providing the services of the process.

The purpose of this generic practice is to ensure that the resources necessary to perform the process as defined by the plan are available when they are needed. Resources include adequate funding, appropriate physical facilities, skilled people, and appropriate tools.

The interpretation of the term “adequate” depends on many factors and can change over time. Inadequate resources may be addressed by increasing resources or by removing requirements, constraints, and commitments.

GP 2.4 Assign Responsibility

Assign responsibility and authority for performing the process, developing the work products, and providing the services of the process.

The purpose of this generic practice is to ensure that there is accountability for performing the process and achieving the specified results throughout the life of the process. The people assigned must have the appropriate authority to perform the assigned responsibilities.

Responsibility can be assigned using detailed job descriptions or in living documents, such as the plan for performing the process. Dynamic assignment of responsibility is another legitimate way to implement this generic practice, as long as the assignment and acceptance of responsibility are ensured throughout the life of the process.

GP 2.5 Train People***Train the people performing or supporting the process as needed.***

The purpose of this generic practice is to ensure that people have the necessary skills and expertise to perform or support the process.

Appropriate training is provided to those who will be performing the work. Overview training is provided to orient people who interact with those who perform the work.

Examples of methods for providing training include self study; self-directed training; self-paced, programmed instruction; formalized on-the-job training; mentoring; and formal and classroom training.

Training supports the successful execution of the process by establishing a common understanding of the process and by imparting the skills and knowledge needed to perform the process.

Refer to the Organizational Training process area for more information about developing skills and knowledge of people so they can perform their roles effectively and efficiently.

GP 2.6 Control Work Products***Place selected work products of the process under appropriate levels of control.***

The purpose of this generic practice is to establish and maintain the integrity of the selected work products of the process (or their descriptions) throughout their useful life.

The selected work products are specifically identified in the plan for performing the process, along with a specification of the appropriate level of control.

Different levels of control are appropriate for different work products and for different points in time. For some work products, it may be sufficient to maintain version control so that the version of the work product in use at a given time, past or present, is known and changes are incorporated in a controlled manner. Version control is usually under the sole control of the work product owner (which can be an individual, group, or team).

Sometimes, it can be critical that work products be placed under formal or baseline configuration management. This type of control includes defining and establishing baselines at predetermined points. These baselines are formally reviewed and approved, and serve as the basis for further development of the designated work products.

Refer to the Configuration Management process area for more information about establishing and maintaining the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits.

Additional levels of control between version control and formal configuration management are possible. An identified work product can be under various levels of control at different points in time.

GP 2.7 Identify and Involve Relevant Stakeholders

Identify and involve the relevant stakeholders of the process as planned.

The purpose of this generic practice is to establish and maintain the expected involvement of relevant stakeholders during the execution of the process.

Involve relevant stakeholders as described in an appropriate plan for stakeholder involvement. Involve stakeholders appropriately in activities such as the following:

- Planning
- Decisions
- Commitments
- Communications
- Coordination
- Reviews
- Appraisals
- Requirements definitions
- Resolution of problems and issues

Refer to the Project Planning process area for more information about planning stakeholder involvement.

The objective of planning stakeholder involvement is to ensure that interactions necessary to the process are accomplished, while not allowing excessive numbers of affected groups and individuals to impede process execution.

Examples of stakeholders that might serve as relevant stakeholders for specific tasks, depending on context, include individuals, teams, management, customers, suppliers, end users, operations and support staff, other projects, and government regulators.

GP 2.8 Monitor and Control the Process***Monitor and control the process against the plan for performing the process and take appropriate corrective action.***

The purpose of this generic practice is to perform the direct day-to-day monitoring and controlling of the process. Appropriate visibility into the process is maintained so that appropriate corrective action can be taken when necessary. Monitoring and controlling the process can involve measuring appropriate attributes of the process or work products produced by the process.

Refer to the Measurement and Analysis process area for more information about developing and sustaining a measurement capability used to support management information needs.

Refer to the Project Monitoring and Control process area for more information about providing an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan.

Subpractices

1. Evaluate actual progress and performance against the plan for performing the process.
The evaluations are of the process, its work products, and its services.
2. Review accomplishments and results of the process against the plan for performing the process.
3. Review activities, status, and results of the process with the immediate level of management responsible for the process and identify issues.
These reviews are intended to provide the immediate level of management with appropriate visibility into the process based on the day-to-day monitoring and controlling of the process, and are supplemented by periodic and event-driven reviews with higher level management as described in GP 2.10.
4. Identify and evaluate the effects of significant deviations from the plan for performing the process.
5. Identify problems in the plan for performing the process and in the execution of the process.
6. Take corrective action when requirements and objectives are not being satisfied, when issues are identified, or when progress differs significantly from the plan for performing the process.

Inherent risks should be considered before any corrective action is taken.

Corrective action can include the following:

- Taking remedial action to repair defective work products or services
- Changing the plan for performing the process
- Adjusting resources, including people, tools, and other resources
- Negotiating changes to the established commitments
- Securing change to the requirements and objectives that must be satisfied
- Terminating the effort

7. Track corrective action to closure.

GP 2.9 Objectively Evaluate Adherence

Objectively evaluate adherence of the process and selected work products against the process description, standards, and procedures, and address noncompliance.

The purpose of this generic practice is to provide credible assurance that the process and selected work products are implemented as planned and adhere to the process description, standards, and procedures. (See the definition of “objectively evaluate” in the glossary.)

Refer to the Process and Product Quality Assurance process area for more information about objectively evaluating processes and work products.

People not directly responsible for managing or performing the activities of the process typically evaluate adherence. In many cases, adherence is evaluated by people in the organization, but external to the process or project, or by people external to the organization. As a result, credible assurance of adherence can be provided even during times when the process is under stress (e.g., when the effort is behind schedule, when the effort is over budget).

GP 2.10 Review Status with Higher Level Management

Review the activities, status, and results of the process with higher level management and resolve issues.

The purpose of this generic practice is to provide higher level management with the appropriate visibility into the process.

Higher level management includes those levels of management in the organization above the immediate level of management responsible for the process. In particular, higher level management can include senior management. These reviews are for managers who provide the policy and overall guidance for the process and not for those who perform the direct day-to-day monitoring and controlling of the process.

Different managers have different needs for information about the process. These reviews help ensure that informed decisions on the planning and performing of the process can be made. Therefore, these reviews are expected to be both periodic and event driven.

APPENDIX G:

Detail on each specific generic practice for CMMI CAPABILITY LEVEL CL-3 DEFINED (SEI 2010)

GP 3.1 Establish a Defined Process

Establish and maintain the description of a defined process.

The purpose of this generic practice is to establish and maintain a description of the process that is tailored from the organization's set of standard processes to address the needs of a specific instantiation. The organization should have standard processes that cover the process area, as well as have guidelines for tailoring these standard processes to meet the needs of a project or organizational function. With a defined process, variability in how the processes are performed across the organization is reduced and process assets, data, and learning can be effectively shared.

Refer to the Integrated Project Management process area for more information about establishing the project's defined process.

Refer to the Organizational Process Definition process area for more information about establishing standard processes and establishing tailoring criteria and guidelines.

The descriptions of the defined processes provide the basis for planning, performing, and managing the activities, work products, and services associated with the process.

Subpractices

1. Select from the organization's set of standard processes those processes that cover the process area and best meet the needs of the project or organizational function.
2. Establish the defined process by tailoring the selected processes according to the organization's tailoring guidelines.
3. Ensure that the organization's process objectives are appropriately addressed in the defined process.
4. Document the defined process and the records of the tailoring.
5. Revise the description of the defined process as necessary.

GP 3.2 Collect Process Related Experiences

Collect process related experiences derived from planning and performing the process to support the future use and improvement of the organization's processes and process assets.

The purpose of this generic practice is to collect process related experiences, including information and artifacts derived from planning and performing the process. Examples of process related experiences include work products, measures, measurement results, lessons learned, and process improvement suggestions. The information and artifacts are collected so that they can be included in the organizational process assets and made available to those who are (or who will be) planning and performing the same or similar processes. The information and artifacts are stored in the organization's measurement repository and the organization's process asset library.

Examples of relevant information include the effort expended for the various activities, defects injected or removed in a particular activity, and lessons learned.

Refer to the Integrated Project Management process area for more information about contributing to organizational process assets.

Refer to the Organizational Process Definition process area for more information about establishing organizational process assets.

APPENDIX H:

Presented to Deloitte's advisors, in the beginning of the TFS defect historic analysis

This presentation uses the term bug with the same meaning as the term defect in the context of the dissertation



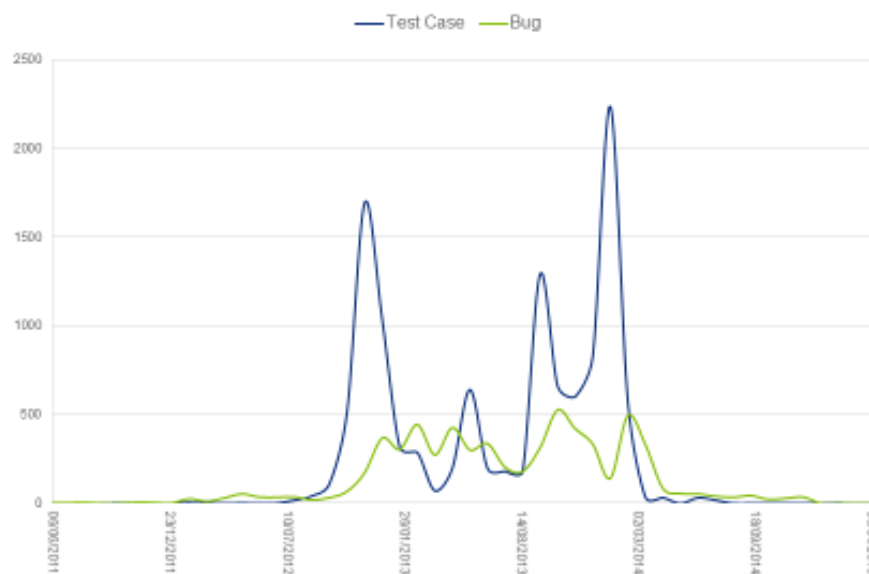
Since early 2013...



Software Defect Classification

2

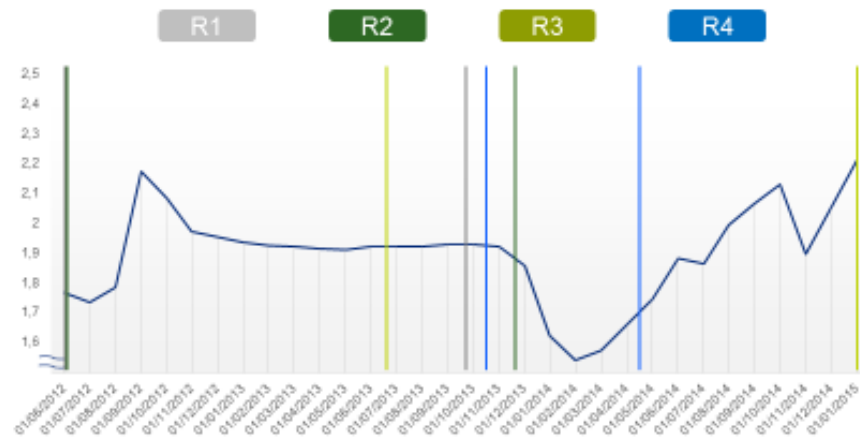
- New items opened daily



Software Defect Classification

3

- Moving Averages – Test Case / Bug

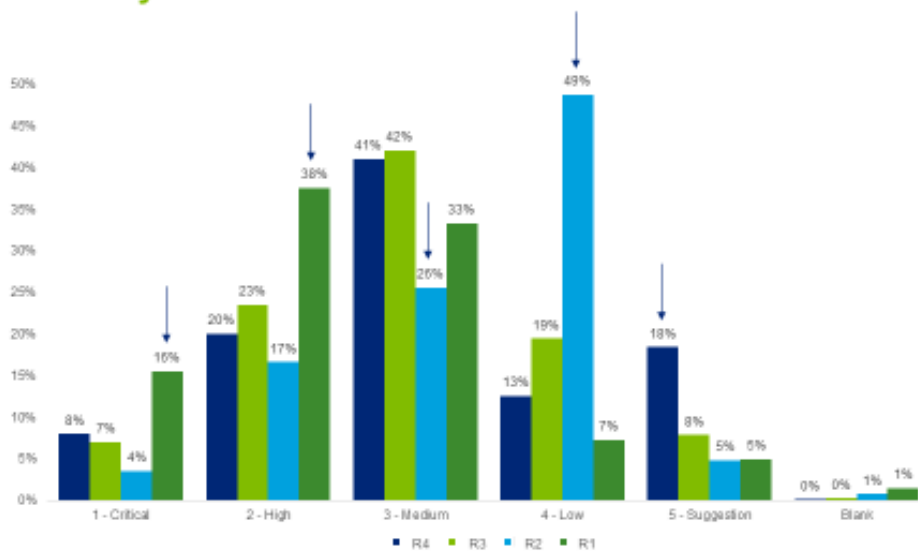


Monthly subset

Software Defect Classification

4

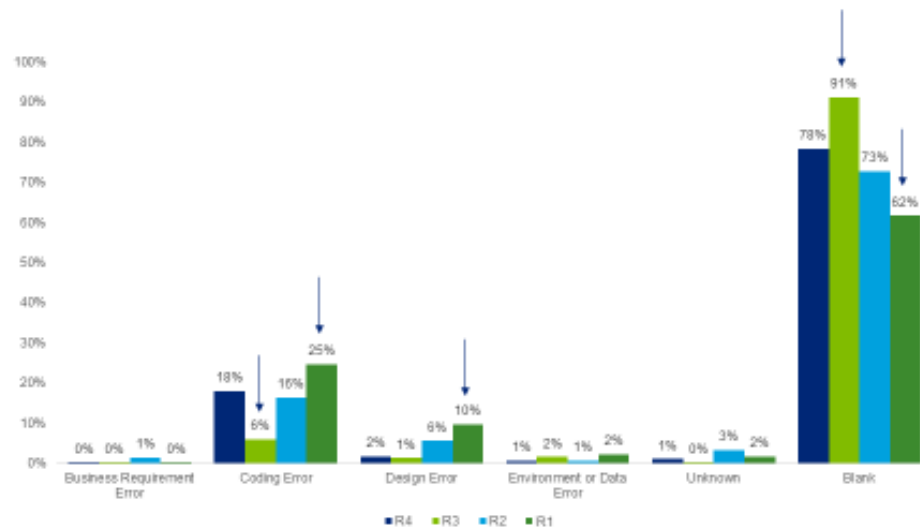
- Severity



Software Defect Classification

5

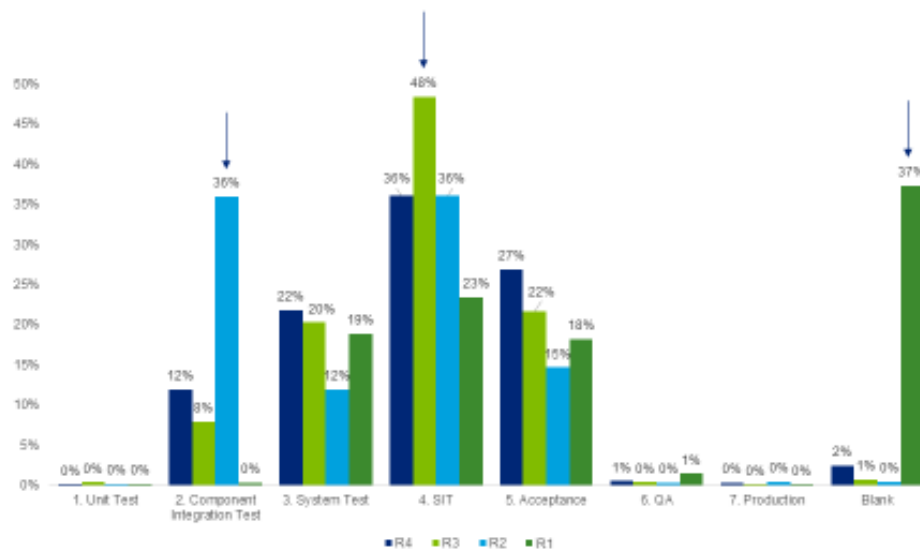
- Root Cause



Software Defect Classification

6

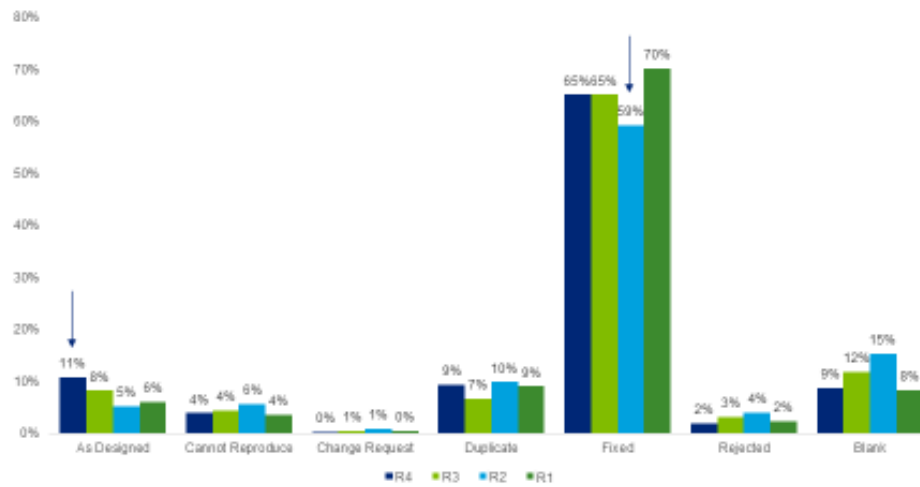
- Found In Environment



Software Defect Classification

7

- Resolved Reason



Software Defect Classification

8

Measures of Central Location and Dispersion...

Measure	Severity	
	Average	Deviation
1 – Critical	8%	5%
2 – High	24%	9%
3 – Medium	35%	8%
4 – Low	22%	19%
5 - Suggestion	9%	6%
Blank	1%	1%

Measure	Root Cause (Error)	
	Average	Deviation
Business Requirement	~0%	1%
Coding	16%	8%
Design	5%	4%
Environment or Data	1%	1%
Unknown	1%	1%
Blank	76%	12%

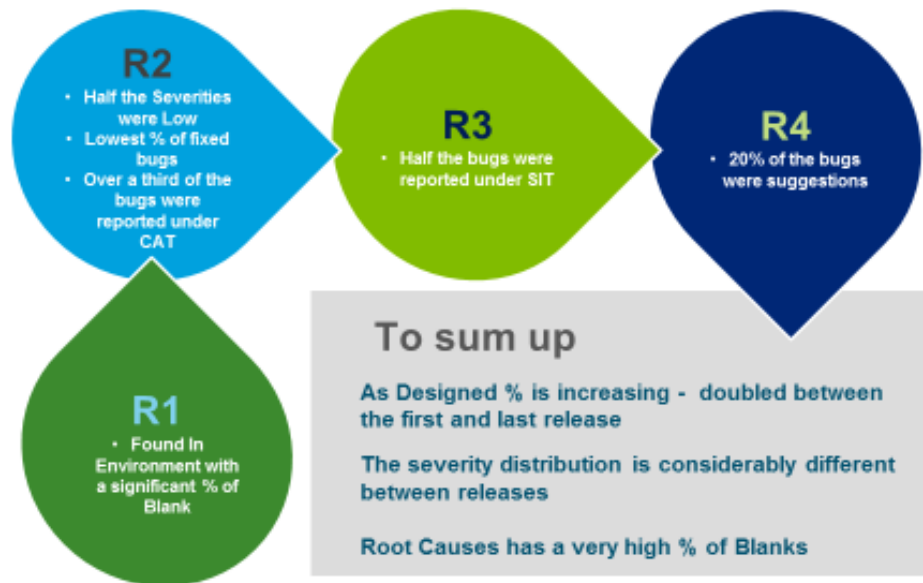
Measure	Environment	
	Average	Deviation
UT	~0%	~0%
CAT	14%	15%
ST	18%	4%
SIT	36%	10%
UAT	20%	5%
QA	1%	1%
Production	~0%	~0%
Blank	10%	18%

Measure	Resolved Reason	
	Average	Deviation
As Designed	8%	3%
Cannot Reproduce	4%	1%
Change Request	1%	~0%
Duplicate	9%	1%
Fixed	65%	4%
Rejected	3%	1%
Blank	11%	2%

Software Defect Classification

9

Wrap Up



APPENDIX I:

Analysis of the survey conducted to Deloitte's professionals in order to inform and create awareness for the subjects under scope of the survey

This presentation uses the term bug with the same meaning as the term defect in the context of the dissertation



Since early 2013...

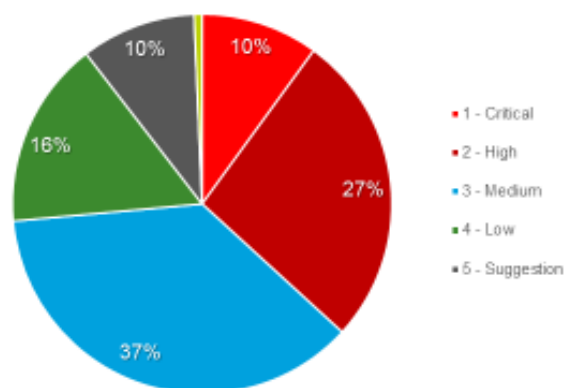


Software Defect Classification

2

Where...

- Over 1/3 of the bugs are rated High and Critical under Severity

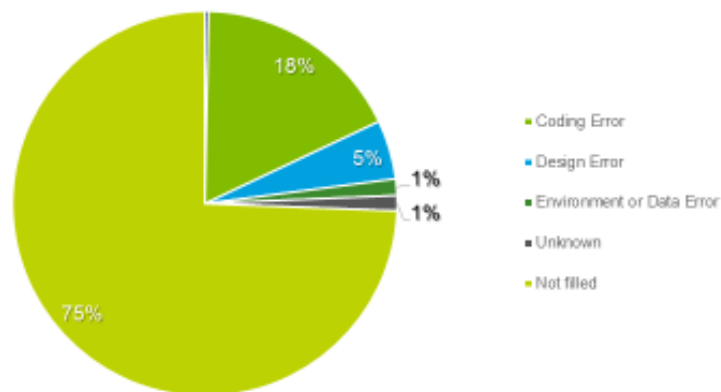


Software Defect Classification

3

And...

- 3/4 of the bugs end without an associated Root Cause
- The most common cause is Coding Error

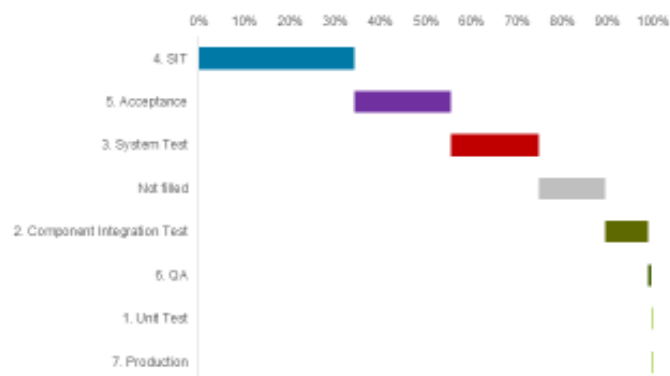


Software Defect Classification

4

Also...

- Since 3/4 of the bugs are found during SIT, UAT and ST, the focus will be set in this phases

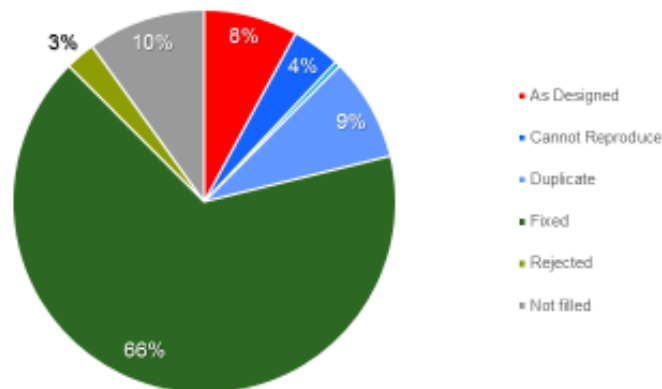


Software Defect Classification

5

And finally...

- 1/3 of the bugs opened are probably “not bugs” (around 2140!)



Software Defect Classification

6

With...

- The Acceptance phase has the least amount of Fixed bugs compared to other testing phases
- This phase also has a significant deviation within releases, and the highest percentage of the As Designed and Duplicates

	3. System Test		4. SIT		5. Acceptance		Total
Measure	Average	Deviation	Average	Deviation	Average	Deviation	% of
As Designed	10%	5%	6%	2%	15%	7%	87%
Cannot Reproduce	6%	6%	5%	1%	5%	2%	92%
Duplicate	5%	4%	8%	3%	13%	7%	69%
Fixed	77%	6%	78%	8%	65%	12%	85%
Rejected	3%	3%	3%	3%	2%	1%	81%

Software Defect Classification

7

Some reasons in the bugs history...

As Designed – “Tester need to make sure that the test case is written correctly. If test case is wrongly written, the test case needs to be updated”

Rejected – “It was not corrected because it is accordingly to the Cargo Product”

Cannot Reproduce – “We cannot reproduce the error. It was probably fixed with full release 4”

As Designed – “This needs to be requested as a change to the FD”

As Designed – “After analysis with the TIA team we identified that the risk category description in TIA is the same as in the print. It seems to be a design flaw”

Duplicated – “Most likely related to indexation issue when choosing manual premium”

Software Defect Classification

8



How many hours are spent handling this kind of bugs?

What are the hidden opportunity costs for that time?



How many people have to break their work flow to look at such issues?

How to reduce the number of opened bugs that “are not bugs”



Software Defect Classification

9

APPENDIX J:

Presented to Deloitte's advisors and to be shared with the other Deloitte's professionals involved in the project and the client in a future moment

This presentation uses the term bug with the same meaning as the term defect in the context of the dissertation



Contents

Scope of the Questionnaire	3
Respondents Characterization	4
Root causes & Resolved Reasons	8
Technical Team Insights	15
Functional Team Insights	21
Improvement Suggestions	25

Scope of the questionnaire

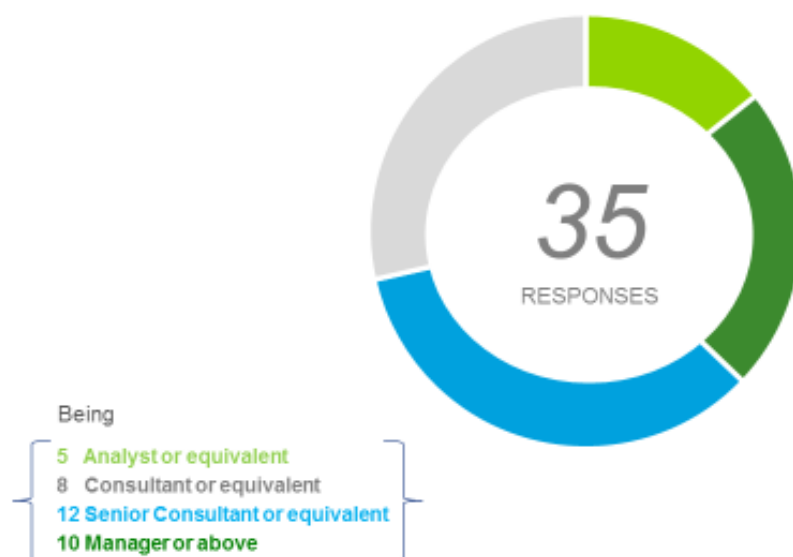


Respondents Characterization

Software Defect Classification

4

Distribution of career levels



Software Defect Classification

5

Distribution of roles

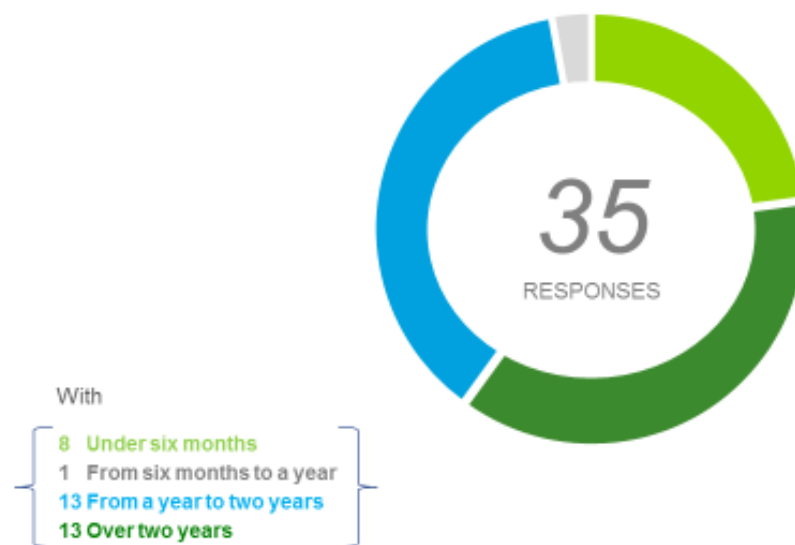


With { - 8 elements from Functional
 - 18 elements from Technical } team having handled bug

Software Defect Classification

6

Permanence on the project



Software Defect Classification

7

Root causes & Resolved Reasons

Software Defect Classification

8

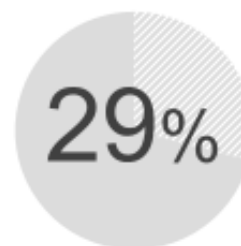
Root Causes for 'Duplicate' bugs



Not checking for similar bugs opened under the same testing phase



Lack of communication



Previous known bugs that were never fixed

Other causes

- Unclear roles and responsibilities within testing teams
- Bugs not properly described / with the same root cause but different functional behaviors

Software Defect Classification

9

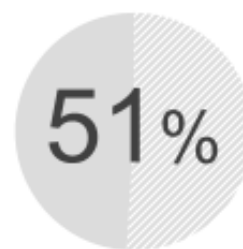
Root Causes for 'As Designed' bugs



Lack of business knowledge



**Incorrect testing
procedure/script**



**Ambiguous/flawed
design**

Other causes

- New future requirement / valid suggestion
- Conflicting requirements
- TIA core limitations
- Functional Design is missing requirements

Software Defect Classification

10

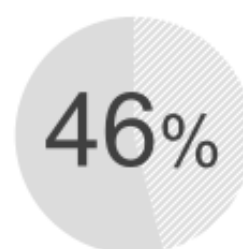
Root Causes for 'Cannot Reproduce' bugs



Unclear/absent repro steps



**No available environments /
different testing conditions
(e. g. new deployments)**



**Description without
the relevant data**

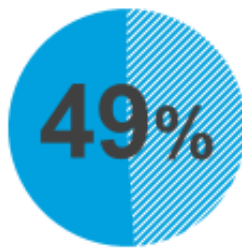
Other causes

- Dependency on service layers
- Technical team does not have enough time

Software Defect Classification

11

Root Causes for 'Reject' bugs



'As Designed'



TIA core limitations



Service layer issues

Other causes

- Bugs that have not been created correctly or do not represent an issue to the system

Software Defect Classification

12

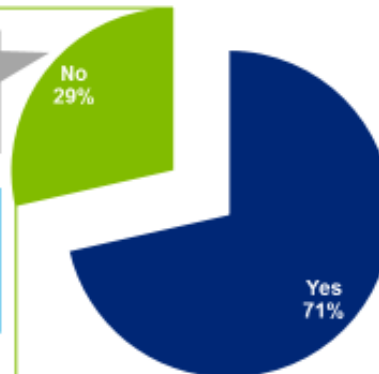
On 'Reject' bugs

To the question

Do you think 'As Designed' and 'Rejected' are being used to address the same nature of causes?

'As Designed' is due to the fact that the functional design and business logic does not cover that option. 'Rejected' is for the bugs that does not apply entirely to the scope

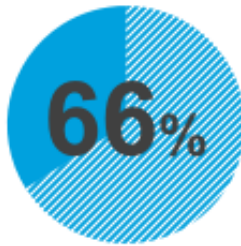
'As Designed' is used when there is a justifiable reason in the functional design. In other hand, the 'Rejected' is used when there's no bug at all but no design to support the claim



Software Defect Classification

13

Bug with no associated reason



Doubt in classifying the bug



**Lack of knowledge on the TFS
usage guidelines**



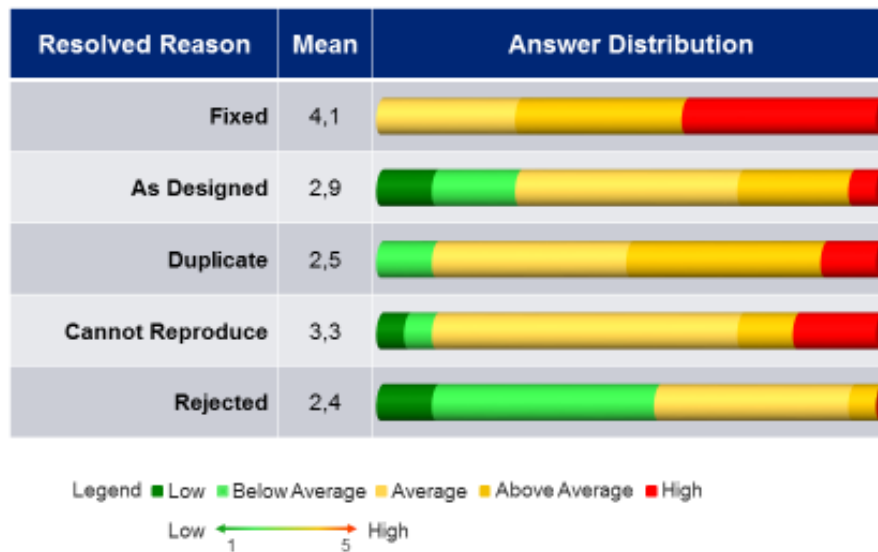
**Other tasks at hand
with higher
importance**

Other causes

- Lack of a solid system for reporting bugs

Technical Team

Work involved in 'fixing'



Software Defect Classification

16

Value delivered by 'bugs that are not bugs'



Reasons

We lose time going after stuff that are duplicated, as designed or that cannot be reproduced, leaving no time to actual bugs

Some times bugs are opened but it does not correspond to the actual development. But it's related to other systems/projects etc

Software Defect Classification

17

On the hassle this bugs cause



Moreover,

Shifting our attention from one task to another, as we do when we're monitoring email while trying to read a report or craft a presentation, disrupts our concentration and saps our focus. Each time we return to our initial task, we use up valuable cognitive resources reorienting ourselves.

Friedman, R. (2014) The Cost of Continuously Checking Email: Harvard Business Review

Software Defect Classification

18

To the question

Do you think functional teams have enough care and do proper research before opening bugs?

8 ✓ 10 ✗

Reasons

Before opening a bug, we need to ensure we know the expected behavior (not only based on Functional Documentation)

If the proper care was made the as design bugs were not opened. Sometimes all that is needed is a clarification with the technical team. Most of the times this is not requested

Software Defect Classification

19

On 'Reject' bugs More Reasons

It mostly depends on how delayed the test phase is. You notice that these bugs tend to appear more when there is more work, as the teams don't have the resources to properly identify these bugs

Sometimes the functional team does not have a clear and full understanding of all the requirements because they did not take part of the design phase. Other times, the functional team does not have the necessary knowledge of the TIA core system

They don't understand the whole flow of what they are testing, and do not study enough the application they are testing. Therefore, they do not have knowledge to make a good triage of these situations

The functional teams sometimes lack on business knowledge and that causes them to open bugs that are not bugs

Software Defect Classification

20

Functional Team

Software Defect Classification

21

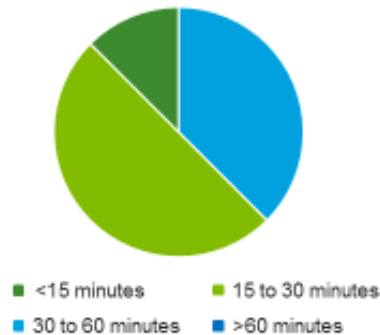
While executing tests

To the question

Do you often have doubts concerning if a certain behavior is a bug or not?



How much time you averagely spent researching the potential bug?



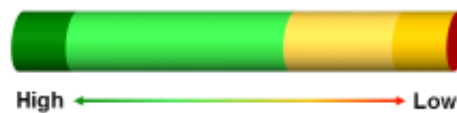
Software Defect Classification

22

While executing tests

To the question

What is averagely the effectiveness of the research?



Which was your main obstacle to decide autonomously if a given behavior should be associated to a bug?

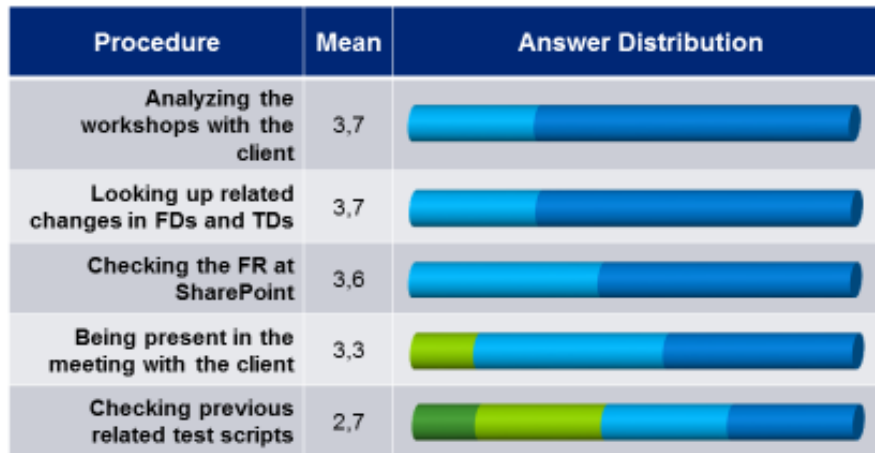
Ambiguous / Incomplete documentation

Lack of knowledge on the application being tested

Software Defect Classification

23

Writing tests – relevance of



Legend ■ Low ■ Medium ■ High ■ Very High

Low 1 4 Very High

Software Defect Classification

24

To everyone

Software Defect Classification

25

What is your perception on...

the complexity of the bug lifecycle and its associated workflow?



the work technical teams have handling bugs that are not bugs?

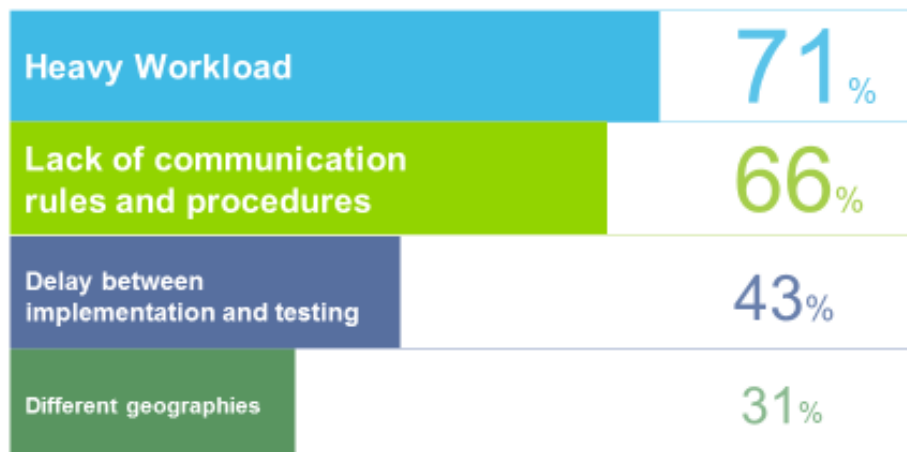
They spend way too much time with this

They tend to reject them immediately

Software Defect Classification

26

On the main obstacles in the interaction between functional and technical teams



Software Defect Classification

27

Improvement suggestions

- 1) *Standardize classification*
- 2) *Review of test cases with technical team*
- 3) *Use team assignment instead of user assignment*

Workshops done by the technical team to explain how the requirements have been implemented and what should be the expected behavior

A more effective filtering before bugs reach technical teams - triage before opening from:

- *a more senior element of the team*
- *a technical element inside the testing team*

Software Defect Classification

28

Improvement suggestions

Monthly Test Academy to refresh concepts and processes
Better knowledge on the solution being tested

First I think that there is a need to create functional design (FD) documents with more and more richer information. Then, for a better bug handling processes is necessary to normalize the process. If the process is well know for all of the teams involved, the time wasted in bug handling for both technical and functional teams it will reduced.

Create user manuals for bug handling with descriptions for classifications, rules, etc. Train people in the bug handling tools and guidelines. Train testers in the major functional requirements and design decisions.

Software Defect Classification

29